



## 功能描述

版本 1.8.0

Web Server 有限责任公司

2024 年 12 月 20 日

---

## 目录

---

<b>1 注释</b>	<b>1</b>
<b>2 一般信息</b>	<b>2</b>
<b>3 功能特性</b>	<b>4</b>
3.1 运行时控制 . . . . .	4
3.1.1 使用信号 . . . . .	5
3.1.2 更改配置 . . . . .	5
3.1.3 轮换日志文件 . . . . .	6
3.2 连接、会话、请求、日志 . . . . .	7
3.2.1 连接处理机制 . . . . .	7
3.2.2 HTTP 请求处理 . . . . .	8
3.2.3 TCP/UDP 会话处理 . . . . .	8
3.2.4 处理请求 . . . . .	9
3.2.5 代理和负载均衡 . . . . .	13
3.2.6 日志记录 . . . . .	16
3.3 内置模块 . . . . .	17
3.3.1 核心模块 . . . . .	17
3.3.2 HTTP 模块 . . . . .	28
3.3.3 流模块 . . . . .	383
3.3.4 邮件模块 . . . . .	465
3.3.5 Google PerfTools 模块 . . . . .	488
3.3.6 WASM 模块 . . . . .	489
3.3.7 核心模块 . . . . .	493
3.3.8 HTTP 模块 . . . . .	493
3.3.9 流模块 . . . . .	495
3.3.10 邮件模块 . . . . .	495
3.3.11 Google PerfTools 模块 . . . . .	496
3.3.12 WASM 模块 . . . . .	496
<b>4 知识产权</b>	<b>497</b>



## 章节 1

---

### 注释

---

本文件包含对 Angie PRO 功能特性的描述。Angie PRO 是一个高效、强大且可扩展的网络服务器。

## 章节 2

---

### 一般信息

---

Angie PRO 是唯一在俄罗斯开发和本地化的商业 Web 服务器。

Web 服务器是一类软件，通过 HTTP 协议为最终用户提供对网络资源的访问。例如，Angie PRO 可用于运营网站、移动应用程序、地铁自助服务终端以及长途列车上的多媒体系统。每当用户打开网站、使用移动应用程序、与地铁里的自助服务终端互动，或甚至与“萨普桑”列车上的多媒体系统交互时，用户的请求都可以由 Angie PRO 处理。

Angie PRO 是：

- **通用 Web 服务器。**用 C 语言编写。
- **L4-L7 负载均衡器。**允许在 TCP/UDP 协议和 HTTP 之间进行负载均衡。
- **代理和缓存服务器。**通过灵活的缓存机制加快 Web 服务的运行。
- **可在所有流行平台上使用。**在 Alpine、Debian、Oracle、RED OS、Rocky 和 Ubuntu 上编译和测试。
- **高性能。**世界上最有效的 Web 服务器之一。

选择 Angie PRO 的理由：

- **与 NGINX OSS 兼容。** Angie PRO 完全兼容 Nginx，允许任何现有的 Nginx 用户在没有重大成本或服务停机的情况下过渡到 Angie PRO。
- **增强的统计信息和实时监控。** Angie PRO 提供完整的实时服务器负载监控，能够根据负载动态管理配置，确保服务的完全可用性。
- **动态配置代理服务器组。**允许通过方便的 REST 接口管理代理服务器组的设置，无需服务中断。
- **缓存元素移除。**提供通过用户友好的 API 移除缓存元素的能力，无需服务停机。

- **对代理服务器的主动健康检查。** 检查“存活状态”，并仅代理到根据指定算法响应的代理服务器组。
- **动态键值存储。** 通过 HTTP API 动态管理 Angie PRO 配置变量。
- **动态 DNS 更新。**
- **会话亲和代理。**
- **具有动态第三方模块的仓库。** Angie PRO 支持大多数 NGINX 第三方模块，并允许无缝安装，确保功能和支持。
- **共享内存区域同步。** 能够在 Angie PRO 集群中使用缓存区域、limit\_req 等功能。
- **在响应头中隐藏或个性化服务器名称。** 能够更改或隐藏 Web 服务器的名称和版本，防止用户看到。

与 Angie PRO 具有类似功能特性的外部软件列表包括 Nginx、Nginx Plus、Apache、Envoy、利用 NGINX 解决方案的产品（OpenResty、Tengine、Cloudflare）以及 Yandex 的云解决方案。

## 章节 3

### 功能特性

#### 3.1 运行时控制

要启动 Angie, 请使用 `systemd` 运行以下命令:

```
$ sudo service angie start
```

建议事先检查配置语法。操作如下:

```
$ sudo angie -t && sudo service angie start
```

要重新加载配置:

```
$ sudo angie -t && sudo service angie reload
```

要停止 Angie:

```
$ sudo service angie stop
```

安装后, 运行以下命令以确保 Angie 正在运行:

```
$ curl localhost:80
```

#### **i** 备注

开源版本的 Angie 的运行方法可能因安装方法而异。

Angie 有一个主进程和多个工作进程。主进程负责读取和评估配置，并维护工作进程。工作进程负责实际请求处理。Angie 使用基于事件的模型和操作系统相关的机制来高效地在工作进程之间分配请求。工作进程的数量在配置文件中定义，可以在给定配置中是固定的，也可以根据可用 CPU 核心的数量自动调整（参见 `worker_processes`）。

### 3.1.1 使用信号

也可以使用信号来控制 Angie。默认情况下，主进程的进程 ID 写入到文件 `/var/run/angie.pid`。可以在配置时或在 `angie.conf` 中使用 `pid` 指令更改此文件名。主进程支持以下信号：

TERM, INT	快速关闭
QUIT	优雅的 关闭
HUP	重新加载配置，更新时区（仅适用于 FreeBSD 和 Linux），使用更新的配置启动新的工作进程，优雅地 关闭旧的工作进程
USR1	重新打开日志文件
USR2	升级可执行文件
WINCH	优雅地 关闭工作进程

单个工作进程也可以使用信号控制，尽管这是可选的。支持的信号有：

TERM, INT	快速关闭
QUIT	优雅的 关闭
USR1	重新打开日志文件
WINCH	用于调试的异常终止（需要启用 <code>debug_points</code> ）

### 3.1.2 更改配置

为了让 Angie 重新读取配置文件，应向主进程发送 HUP 信号。主进程首先检查语法的有效性，然后尝试应用新配置，包括打开新的日志文件和监听套接字。如果应用新配置失败，主进程将回滚更改并继续使用旧配置运行。如果应用成功，主进程将启动新的工作进程，并向旧的工作进程发送消息，要求它们优雅地 关闭。旧的工作进程关闭它们的监听套接字并继续为现有客户端提供服务。在所有客户端服务完成后，旧的工作进程将关闭。

Angie 跟踪每个进程的配置更改。生成号在服务器首次启动时从 1 开始。这些数字随着每次配置重新加载而递增，并在进程标题中可见：

```
$ sudo angie
$ ps aux | grep angie

    angie: master process v1.8.0 #1 [angie]
    angie: worker process #1
```

成功重新加载配置后（无论是否有实际更改），Angie 将为接收到新配置的进程增加生成号：



```
$ sudo angie -s reload
$ ps aux | grep angie

angie: master process v1.8.0 #2 [angie]
angie: worker process #2
```

如果任何来自先前生成的工作进程继续运行, 它们将立即可见:

```
$ ps aux | grep angie

angie: worker process #1
angie: worker process #2
```

#### **i** 备注

不要将配置生成号与“进程编号”混淆; Angie 出于实际目的不使用连续的进程编号。

### 3.1.3 轮换日志文件

要轮换日志文件, 首先重命名文件。然后, 向主进程发送 `USR1` 信号。主进程将重新打开当前所有打开的日志文件, 并将它们分配给工作进程运行的非特权用户。成功重新打开文件后, 主进程关闭所有打开的文件, 并通知工作进程重新打开它们的日志文件。工作进程也将打开新文件并立即关闭旧文件。结果, 旧文件几乎立即可用于后续处理, 例如压缩。

#### 即时可执行文件升级

要升级服务器可执行文件, 首先用新文件替换旧的可执行文件。然后, 向主进程发送 `USR2` 信号。主进程将其当前文件重命名为带有进程 ID 的新文件, 带有 `.oldbin` 后缀, 例如 `/usr/local/angie/logs/angie.pid.oldbin`, 然后启动新的可执行文件, 新的可执行文件将启动新的工作进程。

请注意, 旧主进程不会关闭其监听套接字, 并且可以管理以重新启动其工作进程 (如果需要)。如果新可执行文件未按预期执行, 您可以采取以下操作之一:

- 向旧主进程发送 `HUP` 信号。这将启动新的工作进程, 而无需重新读取配置。然后, 您可以通过向新主进程发送 `QUIT` 信号来优雅地关闭所有新进程。
- 向新主进程发送 `TERM` 信号。它将向其工作进程发送消息, 要求它们立即退出。如果有任何进程未退出, 请发送 `KILL` 信号以强制退出。当新主进程退出时, 旧主进程将自动启动新的工作进程。

如果新主进程退出, 旧主进程将从文件名中移除带有进程 ID 的 `.oldbin` 后缀。

如果升级成功, 向旧主进程发送 `QUIT` 信号, 只有新的进程将保留。

## 命令行选项

-h, -?	显示命令行参数帮助, 然后退出。
--build-env	显示构建环境的辅助信息, 然后退出。
-c <i>file</i>	使用 <i>file</i> 作为配置文件, 而不是默认文件。
-e <i>file</i>	使用 <i>file</i> 作为错误日志文件, 而不是默认文件。特殊值 <code>stderr</code> 指定标准错误输出。
-g <i>directives</i>	应用额外的全局配置指令, 例如: <code>angie -g "pid /var/run/angie.pid; worker_processes `sysctl -n hw.ncpu`";</code>
-m, -M	显示内置 (-m) 或内置和加载 (-M) 模块的列表, 然后退出。
-p <i>prefix</i>	指定 <code>angie</code> 的 <i>prefix</i> 路径 (服务器文件所在的目录; 默认是 <code>/usr/local/angie/</code> )。
-q	仅在设置 <code>-t</code> 或 <code>-T</code> 时显示错误消息; 否则不执行任何操作。
-s <i>signal</i>	向主进程发送信号, 例如 <code>stop</code> 、 <code>quit</code> 、 <code>reopen</code> 、 <code>reload</code> 等。
-t	测试配置文件, 然后退出。Angie 检查配置语法并递归地包含配置中提到的任何文件。
-T	与 <code>-t</code> 相同, 但在递归包含配置中提到的所有文件后, 还将摘要配置输出到标准输出。
-v	显示 Angie 版本, 然后退出。
-V	显示 Angie 版本、编译器版本和使用的构建参数, 然后退出。

## 3.2 连接、会话、请求、日志

### 3.2.1 连接处理机制

Angie 支持多种连接处理方法。特定方法的可用性取决于所使用的平台。在支持多种方法的平台上, Angie 通常会自动选择最有效的方法。然而, 如果需要, 可以使用 `use` 指令明确选择连接处理方法。

以下是可用的连接处理方法:

方法	描述
<code>select</code>	一种标准方法。在没有更高效方法的平台上, 支持模块会自动构建。可以使用 <code>--with-select_module</code> 和 <code>--without-select_module</code> 构建选项来强制启用或禁用此模块的构建。
<code>poll</code>	一种标准方法。在没有更高效方法的平台上, 支持模块会自动构建。可以使用 <code>--with-poll_module</code> 和 <code>--without-poll_module</code> 构建选项来强制启用或禁用此模块的构建。
<code>kqueue</code>	一种在 FreeBSD 4.1+、OpenBSD 2.9+、NetBSD 2.0 和 macOS 上可用的高效方法。
<code>epoll</code>	一种在 Linux 2.6+ 上可用的高效方法。
<code>/dev/poll</code>	一种在 Solaris 7 11/99+、HP/UX 11.22+ (eventport)、IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+ 上可用的高效方法。
<code>eventport</code>	事件端口方法在 Solaris 10+ 上可用。(由于已知问题, 建议使用 <code>/dev/poll</code> 方法。)

### 3.2.2 HTTP 请求处理

HTTP 请求经过一系列阶段, 每个阶段执行特定类型的处理。

Post-read	初始阶段。在此阶段调用 <i>RealIP</i> 模块。
Server-rewrite	在此阶段处理 <code>server</code> 块中 (但在 <code>location</code> 块之外) 定义的 <i>Rewrite</i> 模块指令。
Find-config	一个特殊阶段, 根据请求 URI 选择 <i>location</i> 。
Rewrite	类似于 <code>Server-rewrite</code> 阶段, 但应用于在上一个阶段选择的 <code>location</code> 块中定义的 <i>rewrite</i> 规则。
Post-rewrite	一个特殊阶段, 如果在 <code>Rewrite</code> 阶段中修改了 URI, 则请求被重定向到新位置, 如在 <code>Find-config</code> 阶段。
Preaccess	在此阶段, 标准 <i>Angie</i> 模块如 <i>Limit Req</i> 注册它们的处理程序。
Access	验证客户端请求授权的阶段, 通常通过调用标准 <i>Angie</i> 模块如 <i>Auth Basic</i> 。
Post-access	一个特殊阶段, 处理 <i>satisfy any</i> 指令。
Precontent	在此阶段, 标准模块指令如 <i>try_files</i> 和 <i>mirror</i> 注册它们的处理程序。
Content	通常在此阶段生成响应。多个标准 <i>Angie</i> 模块在此阶段注册它们的处理程序, 包括 <i>Index</i> 、 <i>proxy_pass</i> 、 <i>fastcgi_pass</i> 、 <i>uwsgi_pass</i> 、 <i>scgi_pass</i> 和 <i>grpc_pass</i> 指令也在此处处理。 处理程序按顺序调用, 直到其中一个生成输出。
Log	最终阶段, 执行请求日志记录。目前, 只有 <i>Log</i> 模块在此阶段注册其处理程序用于访问日志。

### 3.2.3 TCP/UDP 会话处理

来自客户端的 TCP/UDP 会话经过一系列阶段, 每个阶段执行特定类型的处理:

Post-accept	接受客户端连接后的初始阶段。在此阶段调用 <i>RealIP</i> 模块。
Pre-access	检查访问权限的初步阶段。在此阶段调用 <i>Set</i> 模块。
Access	在实际数据处理之前限制客户端访问的阶段。在此阶段调用 <i>Access</i> 模块。
SSL	执行 TLS/SSL 终止的阶段。在此阶段调用 <i>SSL</i> 模块。
Preread	将初始数据字节读入 <i>preread buffer</i> 的阶段, 以允许模块如 <i>SSL Preread</i> 在处理前分析数据。
Content	一个必需阶段, 数据在此实际处理, 通常涉及 <i>Return</i> 模块以向客户端发送响应。在此也处理 <i>proxy_pass</i> 指令。
Log	记录客户端会话处理结果的最终阶段。在此阶段调用 <i>Log</i> 模块。

### 3.2.4 处理请求

#### 虚拟服务器选择

最初, 连接在默认服务器的上下文中创建。然后可以在请求处理的以下阶段确定服务器名称, 每个阶段都参与服务器配置的选择:

- 在 SSL 握手时, 根据 SNI 提前确定。
- 在处理请求行之后。
- 在处理 Host 头字段之后。

如果在处理请求行或 Host 头字段之后未确定服务器名称, Angie 将使用空名称作为服务器名称。

在这些阶段中的每一个阶段, 可能会应用不同的服务器配置。因此, 某些指令应谨慎指定:

- 在 `ssl_protocols` 指令的情况下, 协议列表由 OpenSSL 库在服务器配置根据 SNI 请求的名称应用之前设置。因此, 协议应仅为默认服务器指定。
- `client_header_buffer_size` 和 `merge_slashes` 指令在读取请求行之前应用。因此, 这些指令要么使用默认服务器配置, 要么使用 SNI 选择的服务器配置。
- 在处理请求头字段时, `ignore_invalid_headers`、`large_client_header_buffers` 和 `underscores_in_headers` 指令的情况下, 服务器配置还取决于它是否根据请求行或 Host 头字段进行了更新。
- 错误响应使用当前处理请求的服务器中的 `error_page` 指令处理。

#### 基于名称的虚拟服务器

Angie 首先确定哪个服务器应处理请求。考虑一个简单的配置, 其中所有三个虚拟服务器都监听 80 端口:

```
server {  
  
    listen 80;  
    server_name example.org www.example.org;  
    # ...  
}  
  
server {  
  
    listen 80;  
    server_name example.net www.example.net;  
    # ...  
}  
  
server {  
  
    listen 80;
```

```
server_name example.com www.example.com;
# ...
}
```

在此配置中, Angie 仅根据 `Host` 头字段确定哪个服务器应处理请求。如果此头字段的值与任何服务器名称不匹配或请求不包含此头字段, Angie 将请求路由到此端口的默认服务器。在上述配置中, 默认服务器是第一个——这是 Angie 的标准默认行为。还可以使用 `listen` 指令中的 `default_server` 参数显式指定哪个服务器应为默认:

```
server {

    listen 80 default_server;
    server_name example.net www.example.net;
    # ...
}
```

### 备注

请注意, 默认服务器是监听套接字的属性, 而不是服务器名称的属性。

## 国际化名称

国际化域名 (IDNs) 应在 `server_name` 指令中使用 ASCII (Punycode) 表示:

```
server {

    listen 80;
    server_name xn--e1afmkfd.xn--80akhbyknj4f; # ...
    # ...
}
```

## 防止带有未定义服务器名称的请求

如果不应允许没有 `Host` 头字段的请求, 可以定义一个只需丢弃此类请求的服务器:

```
server {

    listen 80;
    server_name "";
    return 444;
}
```

在此配置中, 服务器名称设置为空字符串, 这与没有 `Host` 头字段的请求匹配。然后返回一个特殊的非标准代码 444, 它关闭连接。

## 结合基于名称和基于 IP 的虚拟服务器

让我们看看一个更复杂的配置, 其中一些虚拟服务器监听不同的地址:

```
server {  
  
    listen 192.168.1.1:80;  
    server_name example.org www.example.org;  
    # ...  
}  
  
server {  
  
    listen 192.168.1.1:80;  
    server_name example.net www.example.net;  
    # ...  
}  
  
server {  
  
    listen 192.168.1.2:80;  
    server_name example.com www.example.com;  
    # ...  
}
```

在此配置中, Angie 首先测试请求的 IP 地址和端口与 *server* 块的 *listen* 指令进行匹配。然后测试请求的 Host 头字段与匹配 IP 地址和端口的 *server* 块的 *server\_name* 条目进行匹配。如果未找到服务器名称, 请求将由默认服务器处理。例如, 收到的请求 `www.example.com` 在端口 `192.168.1.1:80` 上将由该端口的默认服务器处理——即第一个服务器——因为 `www.example.com` 未在此端口定义。

如前所述, 默认服务器是监听端口的属性, 可以为不同的端口定义不同的默认服务器:

```
server {  
  
    listen 192.168.1.1:80;  
    server_name example.org www.example.org;  
    # ...  
}  
  
server {  
  
    listen 192.168.1.1:80 default_server;  
    server_name example.net www.example.net;  
    # ...  
}  
  
server {  
  
    listen 192.168.1.2:80 default_server;
```

```
server_name example.com www.example.com;  
# ...  
}
```

## 选择位置

考虑一个简单的 PHP 网站配置:

```
server {  
  
    listen 80;  
    server_name example.org www.example.org;  
    root /data/www;  
  
    location / {  
  
        index index.html index.php;  
    }  
  
    location ~* \.(gif|jpg|png)$ {  
  
        expires 30d;  
    }  
  
    location ~ \.php$ {  
  
        fastcgi_pass localhost:9000;  
        fastcgi_param SCRIPT_FILENAME  
            $document_root$fastcgi_script_name;  
        include fastcgi_params;  
    }  
}
```

Angie 首先搜索由字面字符串给出的最具体的前缀 `location`, 而不管它们的列出顺序。在上述配置中, 唯一的前缀位置是 `location /`, 它匹配任何请求, 并将作为最后的手段使用。然后, Angie 按配置文件中出现的顺序检查由正则表达式定义的位置。第一个匹配的表达式停止搜索, Angie 将使用该 `location`。如果没有正则表达式匹配请求, Angie 将使用之前找到的最具体的前缀 `location`。

### 备注

所有类型的位置仅测试请求行的 URI 部分, 不包括参数。这是因为可以以各种方式指定查询字符串中的参数, 例如:

- `/index.php?user=john&page=1`
- `/index.php?page=1&user=john`

此外, 查询字符串可能包含任意数量的参数:

- `/index.php?page=1&something+else&user=john`

现在让我们看看在上述配置中如何处理请求:

- 请求 `/logo.gif` 首先匹配前缀 `location /`, 然后匹配正则表达式 `.(gif|jpg|png)$`。因此, 它由后者位置处理。使用指令 `root /data/www`, 请求映射到文件 `/data/www/logo.gif`, 并将文件发送给客户端。
- 请求 `/index.php` 也首先匹配前缀 `location /`, 然后匹配正则表达式 `.(php)$`。因此, 它由后者位置处理, 请求被传递到监听 `localhost:9000` 的 FastCGI 服务器。`fastcgi_param` 指令将 FastCGI 参数 `SCRIPT_FILENAME` 设置为 `/data/www/index.php`, FastCGI 服务器执行文件。变量 `$document_root` 设置为 `root` 指令的值, 变量 `$fastcgi_script_name` 设置为请求 URI, 即 `/index.php`。
- 请求 `/about.html` 仅匹配前缀 `location /`, 因此在此位置处理。使用指令 `root /data/www`, 请求映射到文件 `/data/www/about.html`, 并将文件发送给客户端。

处理请求 `/`` 更为复杂。它仅匹配前缀 `location /`, 因此在此位置处理。`index` 指令然后根据其参数和 `root /data/www` 指令测试索引文件的存在。如果文件 `/data/www/index.html` 不存在但文件 `/data/www/index.php` 存在, 该指令执行内部重定向到 `/index.php`, Angie 再次搜索位置, 仿佛请求是由客户端发送的。如前所述, 重定向的请求最终将由 FastCGI 服务器处理。

### 3.2.5 代理和负载均衡

设置 Angie 作为代理服务器是一个常见用途。在此角色中, Angie 接收请求, 将它们转发到代理服务器, 检索来自这些服务器的响应, 并将响应发送回客户端。

一个简单的代理服务器:

```
server {  
  
    location / {  
  
        proxy_pass http://backend:8080;  
    }  
}
```

`proxy_pass` 指令指示 Angie 将客户端请求传递给后台 `backend:8080` (代理服务器)。还有许多额外的 `directives` 可用于进一步配置代理连接。



## FastCGI 代理

Angie 可以用于将请求路由到运行各种框架和编程语言（如 PHP）的 FastCGI 服务器。

与 FastCGI 服务器一起工作的最基本的 Angie 配置涉及使用 `fastcgi_pass` 指令而不是 `proxy_pass` 指令，并使用 `fastcgi_param` 指令设置传递给 FastCGI 服务器的参数。假设 FastCGI 服务器可通过 `localhost:9000` 访问。在 PHP 中，`SCRIPT_FILENAME` 参数用于确定脚本名称，`QUERY_STRING` 参数用于传递请求参数。结果配置如下：

```
server {  
  
    location / {  
  
        fastcgi_pass localhost:9000;  
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;  
        fastcgi_param QUERY_STRING $query_string;  
    }  
  
    location ~ \.(gif|jpg|png)$ {  
  
        root /data/images;  
    }  
}
```

此配置设置了一个服务器，通过 FastCGI 协议将除静态图像之外的所有请求路由到在 `localhost:9000` 上运行的代理服务器。

## WebSocket 代理

要从 HTTP/1.1 升级连接到 WebSocket，使用 HTTP/1.1 中可用的 [协议切换](#) 机制。

不过，这里有一个细微之处：由于 Upgrade 头是一个‘逐跳头’[\(<https://datatracker.ietf.org/doc/html/rfc2616#section-13.5.1>‘\\_](https://datatracker.ietf.org/doc/html/rfc2616#section-13.5.1)，它不会从客户端传递到代理服务器。对于正向代理，客户端可以使用 CONNECT 方法来规避此问题。此方法不适用于反向代理，因为客户端不知道任何代理服务器，并且需要在代理服务器上进行特殊处理。

Angie 实现了一种特殊的操作模式，允许在客户端和代理服务器之间建立隧道，如果代理服务器返回代码 101（协议切换）的响应，并且客户端通过请求中的 Upgrade 头请求协议切换。

如前所述，逐跳头，包括 Upgrade 和 Connection，不会从客户端传递到代理服务器。因此，为了使代理服务器了解客户端想要切换到 WebSocket 协议，这些头必须显式传递：

```
location /chat/ {  
  
    proxy_pass http://backend;  
    proxy_http_version 1.1;  
    proxy_set_header Upgrade $http_upgrade;  
    proxy_set_header Connection "upgrade";  
}
```

更复杂的示例演示了在向代理服务器的请求中 `Connection` 头字段的值如何依赖于客户端请求头中的 `Upgrade` 字段的值:

```
http {  
  
    map $http_upgrade $connection_upgrade {  
  
        default upgrade;  
        '' close;  
    }  
  
    server {  
  
        ...  
  
        location /chat/ {  
  
            proxy_pass http://backend;  
            proxy_http_version 1.1;  
            proxy_set_header Upgrade $http_upgrade;  
            proxy_set_header Connection $connection_upgrade;  
        }  
    }  
}
```

默认情况下, 如果代理服务器在 60 秒内没有传输任何数据, 连接将关闭。可以使用 `proxy_read_timeout` 指令增加此超时。或者, 可以配置代理服务器定期发送 WebSocket ping 帧以重置超时并检查连接是否仍然活动。

## 负载均衡

跨多个应用实例进行负载均衡是一种广泛使用的技术, 以优化资源利用, 最大化吞吐量, 减少延迟, 并确保容错配置。

Angie 可以用作高效的 HTTP 负载均衡器, 将流量分配到多个应用服务器, 从而提高 Web 应用程序的性能、可扩展性和可靠性。

使用 Angie 进行负载均衡的最简单配置可能如下所示:

```
http {  
  
    upstream myapp1 {  
  
        server srv1.example.com;  
        server srv2.example.com;  
        server srv3.example.com;  
    }  
}
```

```
server {  
  
    listen 80;  
  
    location / {  
  
        proxy_pass http://myapp1;  
    }  
}  
}
```

在上述示例中, 三个相同应用的实例在 `srv1` 到 `srv3` 上运行。当负载均衡方法未明确配置时, 默认为轮询。其他支持的负载均衡机制包括: `weight`、`least_conn` 和 `ip_hash`。Angie 中的反向代理实现还支持带内(或被动)服务器健康检查。这些通过在 `upstream` 上下文中的 `server` 块内使用 `max_fails` 和 `fail_timeout` 指令进行配置。

### 3.2.6 日志记录

#### **i** 备注

除了这里列出的选项外, 还可以启用 `debugging log`。

#### Syslog

`error_log` 和 `access_log` 指令支持记录到 `syslog`。以下参数用于配置记录到 `syslog`:

<code>server=address</code>	指定 <code>syslog</code> 服务器的地址。地址可以是域名或 IP 地址, 带有可选端口, 或者是在 <code>"unix:"</code> 前缀后指定的 UNIX 域套接字路径。如果未指定端口, 则使用 UDP 端口 514。如果一个域名解析为多个 IP 地址, 则使用第一个解析的地址。
<code>facility=string</code>	设置 <code>syslog</code> 消息的 <code>facility</code> , 如 RFC 3164 中定义的。可能的 <code>facilities</code> 包括: <code>"kern"</code> 、 <code>"user"</code> 、 <code>"mail"</code> 、 <code>"daemon"</code> 、 <code>"auth"</code> 、 <code>"intern"</code> 、 <code>"lpr"</code> 、 <code>"news"</code> 、 <code>"uucp"</code> 、 <code>"clock"</code> 、 <code>"authpriv"</code> 、 <code>"ftp"</code> 、 <code>"ntp"</code> 、 <code>"audit"</code> 、 <code>"alert"</code> 、 <code>"cron"</code> 、 <code>"local0"</code> 、 <code>"local7"</code> 。默认值是 <code>"local7"</code> 。
<code>severity=string</code>	定义 <code>access_log</code> 的 <code>syslog</code> 消息的严重级别, 如 RFC 3164 中指定的。可能的值与 <code>error_log</code> 指令的第二个参数(级别)相同。默认值是 <code>"info"</code> 。错误消息的严重性由 Angie 确定, 因此在 <code>error_log</code> 指令中忽略此参数。
<code>tag=string</code>	设置 <code>syslog</code> 消息的标签。默认标签是 <code>"angie"</code> 。
<code>nohostname</code>	禁用在 <code>syslog</code> 消息头中添加 <code>hostname</code> 字段。

示例 `syslog` 配置:

```
error_log syslog:server=192.168.1.1 debug;

access_log syslog:server=unix:/var/log/angie.sock,nohostname;
access_log syslog:server=[2001:db8::1]:12345,facility=local7,tag=angie,severity=info combined;
```

#### **i** 备注

为防止泛滥, syslog 条目每秒报告不超过一次。

## 3.3 内置模块

本指南描述了 Angie 的内置模块, 提供了配置示例, 列出了它们的指令和参数, 以及内置变量。

### 3.3.1 核心模块

该模块提供了服务器基本操作所需的基本功能和配置指令, 并处理管理工作进程、配置事件驱动模型, 以及处理传入连接和请求等关键任务。它包括用于设置主进程、错误日志记录和控制服务器低级行为的关键指令。

#### 配置示例

```
user www www;
worker_processes 2;

error_log /var/log/error.log info;

events {

    use kqueue; worker_connections 2048;
}
```

#### 指令

##### accept\_mutex

语法	accept_mutex on   off;
默认值	accept_mutex off;
上下文	events

当启用 `accept_mutex` 时, 工作进程将轮流接受新连接。没有此设置时, 所有工作进程都会被通知新连接, 这可能导致系统资源使用效率低下, 如果新连接的数量较少。

#### 备注

在支持 `EPOLLEXCLUSIVE` 标志的系统上, 或使用 `reuseport` 指令时, 无需启用 `accept_mutex`。

### `accept_mutex_delay`

语法	<code>accept_mutex_delay time;</code>
默认值	<code>accept_mutex_delay 500ms;</code>
上下文	<code>events</code>

如果启用了 `accept_mutex`, 此指令指定工作进程在另一个工作进程已处理新连接时继续接受新连接的最大等待时间。

### `daemon`

语法	<code>daemon on   off;</code>
默认值	<code>daemon on;</code>
上下文	<code>main</code>

决定 Angie 是否作为守护进程运行。这主要用于开发过程。

### `debug_connection`

语法	<code>debug_connection address   CIDR   unix;;</code>
默认值	—
上下文	<code>events</code>

为特定客户端连接启用调试日志。其他连接将使用 `error_log` 指令设置的日志级别。可以通过 IPv4 或 IPv6 地址、网络或主机名指定连接。对于使用 UNIX 域套接字的连接, 使用 `unix:` 参数启用调试日志。

```
events {  
  
    debug_connection 127.0.0.1;  
    debug_connection localhost;  
    debug_connection 192.0.2.0/24;  
    debug_connection ::1;
```

```
debug_connection 2001:0db8::/32;  
debug_connection unix;;  
# ...  
}
```

### 重要

为使此指令生效, Angie 必须使用启用 调试日志构建。

## debug\_points

语法	<code>debug_points abort   stop;</code>
默认值	—
上下文	main

此指令用于调试。

当出现内部错误时, 例如工作进程重启期间的套接字泄漏, 启用 `debug_points` 将创建一个核心文件 (`abort`) 或停止进程 (`stop`) 以便使用系统调试器进行进一步分析。

## env

语法	<code>env variable[=<i>value</i>];</code>
默认值	<code>env TZ;</code>
上下文	main

默认情况下, Angie 会删除从父进程继承的所有环境变量, 除了 `TZ` 变量。此指令允许您保留一些继承的变量, 修改其值, 或创建新的环境变量。

这些变量随后:

- 在可执行文件的实时升级 时继承
- 被 *Perl* 模块使用
- 可供工作进程使用

请注意, 以这种方式控制系统库可能并不总是有效, 因为库通常只在初始化期间检查变量, 这发生在此指令生效之前。`TZ` 变量始终继承并可供 *Perl* 模块使用, 除非明确配置为否则。

示例:

```
env MALLOC_OPTIONS;  
env PERL5LIB=/data/site/modules;  
env OPENSSL_ALLOW_PROXY_CERTS=1;
```

**i** 备注

ANGIE 环境变量由 Angie 内部使用, 用户不应直接设置。

**error\_log**

语法	<code>error_log file [level];</code>
默认值	<code>error_log logs/error.log error;</code> (路径取决于 <code>--error-log-path</code> 构建选项)
上下文	main, http, mail, stream, server, location

配置日志记录, 允许在同一配置级别指定多个日志。如果在 main 配置级别未显式定义日志文件, 将使用默认文件。

第一个参数指定用于存储日志的文件。特殊值 `stderr` 选择标准错误流。要配置日志记录到 `syslog`, 使用 `"syslog:"` 前缀。要记录到 循环内存缓冲区, 使用 `"memory:"` 前缀后跟缓冲区大小; 这通常用于调试。

第二个参数设置日志级别, 可以是以下之一: `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert`, 或 `emerg`。这些级别按照严重性递增顺序排列。设置日志级别将捕获等于和更高严重性的消息:

设置	捕获的级别
<code>debug</code>	<code>debug, info, notice, warn, error, crit, alert, emerg</code>
<code>info</code>	<code>info, notice, warn, error, crit, alert, emerg</code>
<code>notice</code>	<code>notice, warn, error, crit, alert, emerg</code>
<code>warn</code>	<code>warn, error, crit, alert, emerg</code>
<code>error</code>	<code>error, crit, alert, emerg</code>
<code>crit</code>	<code>crit, alert, emerg</code>
<code>alert</code>	<code>alert, emerg</code>
<code>emerg</code>	<code>emerg</code>

如果省略此参数, 则使用 `error` 作为默认日志级别。

**ii** 重要

为使 `debug` 日志级别生效, Angie 必须使用启用 调试日志构建。

## events

语法	<code>events { ... };</code>
默认值	—
上下文	main

提供用于影响连接处理的指令的配置文件上下文。

## include

语法	<code>include file   mask;</code>
默认值	—
上下文	any

将另一个文件或符合指定 *mask* 的文件包含到配置中。包含的文件必须包含语法正确的指令和块。

示例:

```
include mime.types;  
include vhosts/*.conf;
```

## load\_module

语法	<code>load_module file;</code>
默认值	—
上下文	main

从指定文件加载动态模块。如果提供了相对路径, 它将根据 `--prefix` 构建选项进行解释。要验证路径:

```
$ sudo angie -V
```

示例:

```
load_module modules/nginx_mail_module.so;
```



## lock\_file

语法	<code>lock_file file;</code>
默认值	<code>lock_file logs/angie.lock;</code> (路径取决于 <code>--lock-path</code> 构建选项)
上下文	<code>main</code>

Angie 使用锁定机制实现 `accept_mutex` 并串行访问共享内存。在大多数系统上, 锁是通过原子操作管理的, 因此不需要此指令。然而, 在某些系统上, 使用替代的 `lock file` 机制。此指令设置锁文件名称的前缀。

## master\_process

语法	<code>master_process on   off;</code>
默认值	<code>master_process on;</code>
上下文	<code>main</code>

决定是否启动工作进程。此指令是为 Angie 开发者准备的。

## multi\_accept

语法	<code>multi_accept on   off;</code>
默认值	<code>multi_accept off;</code>
上下文	<code>events</code>

<code>on</code>	工作进程将同时接受所有新连接。
<code>off</code>	工作进程将一次接受一个新连接。

### 备注

如果使用 `kqueue` 连接处理方法, 则忽略此指令, 因为它提供了准备接受的新连接数量。

## pcre\_jit

语法	<code>pcre_jit on   off;</code>
默认值	<code>pcre_jit off;</code>
上下文	main

为在配置解析时已知的正则表达式启用或禁用“即时编译”（PCRE JIT）。

PCRE JIT 可以显著加速正则表达式处理。

### 重要

JIT 在从版本 8.20 开始的 PCRE 库中可用，前提是它们是使用 `--enable-jit` 配置选项构建的。当 Angie 使用 PCRE 库构建时 (`--with-pcre=`)，使用 `--with-pcre-jit` 选项启用 JIT 支持。

## pid

语法	<code>pid file   off;</code>
默认值	<code>pid logs/angie.pid;</code> （路径取决于 <code>--pid-path</code> 构建选项）
上下文	main

指定将存储 Angie 主进程 ID 的 *file*。该文件是原子创建的，确保其内容始终正确。`off` 设置禁用此文件的创建。

### 备注

如果在重新配置期间修改 *file* 设置但指向前一个 PID 文件的符号链接，则文件不会被重新创建。

## ssl\_engine

语法	<code>ssl_engine device;</code>
默认值	—
上下文	main

指定硬件 SSL 加速器的名称。

## thread\_pool

语法	<code>thread_pool name threads=number [max_queue=number];</code>
默认值	<code>thread_pool default threads=32 max_queue=65536;</code>
上下文	main

定义用于多线程文件读取和发送的线程池的 *name* 和参数, 不阻塞工作进程。

*threads* 参数指定池中的线程数。

如果所有线程都忙, 新任务将等待在队列中。*max\_queue* 参数限制队列中可以等待的任务数量。默认情况下, 队列可以容纳多达 65536 个任务。当队列溢出时, 新任务将以错误结束。

## timer\_resolution

语法	<code>timer_resolution interval;</code>
默认值	—
上下文	main

降低工作进程中的定时器分辨率, 从而减少 `gettimeofday()` 系统调用的频率。默认情况下, 每次查询内核事件时都调用 `gettimeofday()`。降低分辨率后, 它仅在指定的间隔内调用一次。

示例:

```
timer_resolution 100ms;
```

间隔的内部实现取决于使用的方法:

- 如果使用 *kqueue*, 则为 `EVFILT_TIMER` 过滤器。
- 如果使用 *eventport*, 则为 `timer_create()` 函数。
- 否则为 `setitimer()` 函数。

## use

语法	<code>use method;</code>
默认值	—
上下文	events

指定要使用的连接处理 *method*。通常不需要显式指定它, 因为 Angie 默认使用最有效的方法。

## user

语法	<code>user user [group];</code>
默认值	<code>user &lt;--user build option&gt; &lt;--group build option&gt;;</code>
上下文	main

定义工作进程的用户和组凭证（另请参见 构建选项）。如果仅设置用户，则指定的用户名也用于组。

## worker\_aio\_requests

语法	<code>worker_aio_requests number;</code>
默认值	<code>worker_aio_requests 32;</code>
上下文	events

在使用和`epoll`连接处理方法时，设置单个工作进程的最大未完成异步 I/O 操作 `number`。

## worker\_connections

语法	<code>worker_connections number;</code>
默认值	<code>worker_connections 512;</code>
上下文	events

设置工作进程可以打开的最大同时连接数。

请注意，此数字包括所有连接，例如与代理服务器的连接，而不仅仅是客户端连接。此外，同时连接的实际数量不能超过系统的打开文件限制，这可以通过`worker_rlimit_nofile`调整。

语法	<code>worker_cpu_affinity cpumask ...;</code> <code>worker_cpu_affinity auto [cpumask];</code>
默认值	—
上下文	main

将工作进程绑定到特定的 CPU 集。每个 CPU 集由表示允许 CPU 的位掩码表示。每个工作进程应定义单独的集。默认情况下，工作进程不绑定到任何特定的 CPU。

例如：

```
worker_processes 4;
worker_cpu_affinity 0001 0010 0100 1000;
```

此配置将每个工作进程绑定到一个单独的 CPU。

或者:

```
worker_processes 2;  
worker_cpu_affinity 0101 1010;
```

这将第一个工作进程绑定到 CPU0 和 CPU2, 第二个工作进程绑定到 CPU1 和 CPU3。此设置适用于超线程。

特殊值 `auto` 允许自动将工作进程绑定到可用 CPU:

```
worker_processes auto;  
worker_cpu_affinity auto;
```

可选的掩码参数可用于限制可用于自动绑定的 CPU:

```
worker_cpu_affinity auto 01010101;
```

### 重要

此指令仅在 FreeBSD 和 Linux 上可用。

## worker\_priority

语法	<code>worker_priority number;</code>
默认值	<code>worker_priority 0;</code>
上下文	<code>main</code>

定义工作进程的调度优先级, 类似于 `nice` 命令: 负的 `number` 表示更高的优先级。允许的范围通常从 -20 到 20。

示例:

```
worker_priority -10;
```

## worker\_processes

语法	<code>worker_processes number   auto;</code>
默认值	<code>worker_processes 1;</code>
上下文	<code>main</code>

定义工作进程的数量。

最佳值取决于多种因素, 包括 CPU 核心数、硬盘驱动器数量和负载模式。如果不确定, 建议从可用 CPU 核心数开始。值 `auto` 尝试自动检测最佳的工作进程数量。

### `worker_rlimit_core`

语法	<code>worker_rlimit_core size;</code>
默认值	—
上下文	<code>main</code>

更改工作进程的核心文件最大大小限制 (`RLIMIT_CORE`)。这样可以在不重启主进程的情况下增加限制。

### `worker_rlimit_nofile`

语法	<code>worker_rlimit_nofile number;</code>
默认值	—
上下文	<code>main</code>

更改工作进程的最大打开文件数限制 (`RLIMIT_NOFILE`)。这样可以在不重启主进程的情况下增加限制。

### `worker_shutdown_timeout`

语法	<code>worker_shutdown_timeout time;</code>
默认值	—
上下文	<code>main</code>

配置工作进程优雅关机的超时时间。一旦 `time` 到期, Angie 将尝试关闭所有活动连接以完成关机过程。

优雅关机是通过向主进程发送 `QUIT` 信号来启动的, 告诉工作进程停止接受新连接, 同时允许现有连接完成。工作进程继续处理活动请求直到完成, 然后它们干净地退出。如果连接持续打开超过 `worker_shutdown_timeout`, Angie 将强制关闭这些连接以完成关机。

### `working_directory`

语法	<code>working_directory directory;</code>
默认值	—
上下文	<code>main</code>

定义工作进程的当前工作目录。这主要用于写入核心文件, 因此工作进程必须对指定的目录具有写权限。

### 3.3.2 HTTP 模块

#### 访问控制

该模块根据客户端的 IP 地址或网络控制对服务器资源的访问。它允许允许或阻止特定的 IP 地址、IP 范围或 UNIX 域套接字, 通过限制对网站或应用程序敏感区域的访问来增强安全性。

访问还可以通过使用 *Auth Basic* 模块的密码或基于使用 *Auth Request* 模块的子请求结果来限制。要同时应用地址和密码限制, 请使用 *satisfy* 指令。

#### 配置示例

```
location / {  
  
    deny 192.168.1.1;  
    allow 192.168.1.0/24;  
    allow 10.1.1.0/16;  
    allow 2001:0db8::/32;  
    deny all;  
}
```

规则按顺序进行评估, 直到找到匹配项。在此示例中, 仅允许 IPv4 网络 10.1.1.0/16 和 192.168.1.0/24 的访问, 排除特定地址 192.168.1.1, 以及 IPv6 网络 2001:0db8::/32 的访问。当有多个规则时, 建议使用 *Geo* 模块的变量。

#### 指令

##### allow

Syntax	<code>allow address   CIDR   unix:   all;</code>
默认值	—
Context	http, server, location, limit_except

允许指定网络或地址的访问。特殊值 `all` 表示所有客户端 IP。

Added in version 1.5.1: 特殊值 `unix:` 允许对任何 UNIX 域套接字的访问。

## deny

Syntax	<code>deny address   CIDR   unix:   all;</code>
默认值	—
Context	http, server, location, limit_except

拒绝指定网络或地址的访问。特殊值 `all` 表示所有客户端 IP。

Added in version 1.5.1: 特殊值 `unix:` 拒绝对任何 UNIX 域套接字的访问。

## ACME

提供使用 ACME 协议的自动证书获取功能。

当从源代码构建时, 默认不会构建此模块; 需要使用 `--with-http_acme_module` 构建选项启用。

在来自我们的软件包中的包和镜像中, 模块已包含在构建中。

在配置中启用证书检索的步骤:

1. **定义一个 ACME 客户端** 在 `http` 块中, 用 `acme_client` 指令设置一个唯一的客户端名称和其他选项; 可以配置多个 ACME 客户端。
2. **指定要请求证书的域名:** 将为所有 `server` 块中 `server_name` 指令列出的域名颁发单个证书, 前提是这些块的 `acme` 指令引用了单个 ACME 客户端。
3. **确保接受 ACME 挑战:** 通过保持 80 端口开放; 模块将处理其余部分。目前, Angie 仅支持通过 HTTP 进行的域名验证, 这需要响应来自证书颁发机构 (CA) 的特殊请求, 称为 ACME 挑战。
4. **使用新获取的证书和密钥配置 SSL:** 模块将其证书和密钥作为内置变量暴露, 您可以在配置中使用它们来填充 `ssl_certificate` 和 `ssl_certificate_key`。

## 实现细节

客户端密钥和证书使用 PEM 编码存储在由 `--http-acme-client-path` 参数设置的目录下的相应子目录中:

```
$ ls /var/lib/angie/acme/example/  
  
account.key  certificate.pem  private.key
```

ACME 客户端需要与 CA 服务器的账户。为了创建和管理账户, 客户端使用私人密钥 (`account.key`); 如果尚未拥有, 将在启动时创建密钥。然后, 客户端使用它在服务器上注册一个账户。

### 备注

如果您已有现有账户密钥, 请在启动前放置到客户端的子目录中以重用账户。



ACME 客户端还维护一个专用密钥 (`private.key`) 用于证书签名请求 (CSR); 如果需要, 此证书密钥也会在启动时自动创建。

在启动时, 客户端请求一个证书 (如果尚未获得), 为其管理的所有域名签署并发送 CSR 到 CA 服务器。服务器通过 HTTP 验证域名所有权并颁发证书, 客户端在本地存储 (`certificate.pem`)。

当证书接近计划的续订或域名列表更改时, 客户端会签署并发送另一个 CSR 到 CA 服务器。再次, 服务器验证所有权并颁发新证书, 客户端在本地安装, 替换以前的证书。

## 配置示例

这里, 一个名为 `example` 的 ACME 客户端管理 `example.com` 和 `www.example.com` 域名。证书及其密钥通过前缀变量 `$acme_cert_<name>` 和 `$acme_cert_key_<name>` 暴露。它们存储各自的文件内容, 反过来用于 `ssl_certificate` 和 `ssl_certificate_key`:

```
http {  
  
    resolver 127.0.0.53; # 'acme_client'指令需要  
  
    acme_client example https://acme-v02.api.letsencrypt.org/directory;  
  
    server {  
  
        listen          80;          # 可以出现在另一个服务器块中,  
                                     # 使用不同的域名列表  
                                     # 甚至不使用  
  
        listen          443 ssl;  
  
        server_name     example.com www.example.com;  
        acme            example;  
  
        ssl_certificate  $acme_cert_example;  
        ssl_certificate_key $acme_cert_key_example;  
    }  
}
```

如前所述, 80 端口必须开放以接受通过 HTTP 的 ACME 挑战。然而, 如上例所示, `listen` 指令可能出现在单独的 `server` 块中。如果没有这样的块存在, 您甚至可以将新的块限制为仅接收 ACME 挑战:

```
server {  
    listen 80;  
    return 444; # 无响应, 连接被关闭  
}
```

为什么这会起作用? 模块在读取头部后, 但在选择虚拟服务器之前或任何 `rewrite` 和 `location` 指令被处理之前, 拦截请求 `/.well-known/acme-challenge/<TOKEN>`。如果 `TOKEN` 与特定挑战的预期匹配, 则处理此类拦截请求。没有实际的目录访问发生; 请求由模块完全处理。

## 指令

### acme

Syntax	<code>acme name;</code>
Default	—
Context	server

对于所有引用 `acme_client` 名为 `name` 的 `server` 块中的 `server_name` 指令指定的所有域, 将获得单个证书; 如果 `server_name` 配置更改, 证书将更新以反映更改。

#### **i** 备注

目前, 不支持通配符域和使用正则表达式指定的域, 将被忽略。

此指令可以多次指定以加载不同类型的证书, 例如, RSA 和 ECDSA:

```
server {  
  
    listen 443 ssl;  
    server_name example.com www.example.com;  
  
    ssl_certificate $acme_cert_rsa;  
    ssl_certificate_key $acme_cert_key_rsa;  
  
    ssl_certificate $acme_cert_ecdsa;  
    ssl_certificate_key $acme_cert_key_ecdsa;  
  
    acme rsa;  
    acme ecdsa;  
}
```

### acme\_client

Syntax	<code>acme_client name uri [enabled=on   off] [key_type=type] [key_bits=number] [email=email] [renew_before_expiry=time] [retry_after_error=off time];</code>
Default	—
Context	http

在全局范围内定义一个唯一的 ACME 客户端 'name'。它必须对文件目录有效, 并将被视为不区分大小写。

第二个必填参数是 ACME 目录的 ‘uri’。例如, 让我们加密 ACME 目录的 URI 列出为 `https://acme-v02.api.letsencrypt.org/directory`。

为了使此指令生效, 必须在相同上下文中配置 `resolver`。

**i 备注**

出于测试目的, 证书颁发机构通常提供单独的测试环境。例如, 让我们加密测试环境 当前为 `https://acme-staging-v02.api.letsencrypt.org/directory`。

<code>enabled</code>	启用或禁用客户端; 这很有用, 例如, 暂时禁用客户端而不将其从配置中移除。 默认: <code>on</code> 。
<code>key_type</code>	证书的私钥算法类型。有效值: <code>rsa</code> , <code>ecdsa</code> 。 默认: <code>ecdsa</code> 。
<code>key_bits</code>	证书密钥的位数。默认: <code>ecdsa</code> 为 256, <code>rsa</code> 为 2048。
<code>email</code>	用于反馈的可选电子邮件地址; 在 CA 服务器上创建账户时使用。
<code>max_cert_size</code>	指定新证书文件的最大允许大小 (以字节为单位), 以在共享内存中为新证书保留空间。请求的域越多, 需要的空间越多。 如果启动时已存在证书但其大小超过 <code>max_cert_size</code> 的值, 则 <code>max_cert_size</code> 值会动态增加以匹配现有证书文件的大小。 如果更新的证书大小超过 <code>max_cert_size</code> , 则更新将失败并报错。 默认: 8192。
<code>renew_before_exp</code>	证书到期前的 时间, 在此时间点应开始证书更新。 默认: 30d。
<code>retry_after_erro</code>	在无法获取证书时重试的 时间。如果设置为 <code>off</code> , 客户端不会在失败后重试获取证书。 默认: 2h。

**acme\_client\_path**

Syntax	<code>acme_client_path path;</code>
Default	—
Context	http

覆盖用于存储证书和密钥的目录的 `path`, 在构建时用 `--http-acme-client-path` 构建选项指定。

## 内置变量

`$acme_cert_<name>`

由此 *name* 的客户端获得的最后一个证书文件（如果有）的内容。

`$acme_cert_key_<name>`

此 *name* 的客户端使用的证书密钥文件的内容。

### 重要

证书文件仅在 ACME 客户端获得至少一个证书后可用，而密钥文件在启动后立即可用。

## 附加模块

该模块是一个过滤器，用于在响应之前和之后添加文本。

当从源代码构建时，该模块默认不被构建；它需要通过 `--with-http_addition_module` 构建选项来启用。

在来自我们的仓库的包和镜像中，该模块已包含在构建中。

## 配置示例

```
location / {
    add_before_body /before_action;
    add_after_body /after_action;
}
```

## 指令

### add\_before\_body

语法	<code>add_before_body uri;</code>
默认值	—
上下文	http, server, location

在响应体之前添加处理给定子请求返回的文本。参数为空字符串 ("" ) 则取消继承自先前配置级别的添加。

## add\_after\_body

语法	<code>add_after_body uri;</code>
默认值	—
上下文	http, server, location

在响应体之后添加处理给定子请求返回的文本。参数为空字符串 ("" ) 则取消继承自先前配置级别的添加。

## addition\_types

语法	<code>addition_types mime-type ...;</code>
默认值	<code>addition_types text/html;</code>
上下文	http, server, location

允许在具有指定 MIME 类型的响应中添加文本, 除了"text/html" 之外。特殊值"\*" 匹配任何 MIME 类型。

## API

该模块提供 HTTP RESTful 接口, 以 JSON 格式访问有关 Web 服务器实例的基本信息, 以及客户端连接、共享内存区域、DNS 查询、HTTP 请求、HTTP 响应缓存、流模块的 TCP/UDP 会话, 以及连接限制、连接限制、请求限制和上游模块区域的指标。

API 接受 GET 和 HEAD HTTP 请求; 其他请求方法将导致错误:

```
{
  "error": "MethodNotAllowed",
  "description": "The POST method is not allowed for the requested API element \"/\"."
}
```

Angie PRO API 具有动态配置功能, 允许在不重新加载配置或重启 Angie 的情况下更新设置; 目前, 它支持配置上游中的各个后端节点。

## 指令

### api

语法	<code>api path;</code>
默认值	—
上下文	location

在 `location` 中启用 HTTP RESTful 接口。

`path` 参数是必需的, 类似于 `alias` 指令, 但它作用于 API 树, 而不是文件系统层级。

当前缀位置指定时:

```
location /stats/ {
    api /status/http/server_zones/;
}
```

请求 URI 中与前缀位置 `/stats/` 匹配的部分将被指令参数中的路径 `/status/http/server_zones/` 替换。例如, 对于请求 `/stats/foo/`, 将访问 API 元素 `/status/http/server_zones/foo/`。

还可以使用变量: `api /status/$module/server_zones/$name/`, 该指令也可以在正则表达式位置中指定:

```
location ~~/api/([^/]+)/(.*)$ {
    api /status/http/$1_zones/$2;
}
```

这里, 类似于 `alias`, 参数定义了 API 元素的完整路径。例如, 对于 `/api/location/bar/data/`, 以下位置变量将被填充:

```
$1 = "location"
$2 = "bar/data/"
```

经过插值后, 生成的 API 请求路径为 `/status/http/location_zones/bar/data`。

### 备注

在 Angie PRO 中, 可以将动态配置 API 与反映当前状态的不可变指标 API 分离:

```
location /config/ {
    api /config/;
}

location /status/ {
    api /status/;
}
```

总体上, 它用于精确配置 API 的访问权限, 例如:

```
location /status/ {
    api /status/;

    allow 127.0.0.1;
    deny all;
}
```

或者:

```
location /blog/requests/ {
    api /status/http/server_zones/blog/requests/;

    auth_basic          "blog";
    auth_basic_user_file conf/htpasswd;
}
```

### api\_config\_files

语法	api_config_files on   off;
默认值	off
上下文	location

启用或禁用 `config_files` 对象, 该对象列举服务器实例当前加载的所有 Angie 配置文件的内容, 位于 `/status/angie/` API 部分。例如, 使用以下配置:

```
location /status/ {
    api /status/;
    api_config_files on;
}
```

查询 `/status/angie/` 将返回类似如下内容:

```
{
  "version": "1.8.0",
  "address": "192.168.16.5",
  "generation": 1,
  "load_time": "2024-12-19T12:58:39.789Z",
  "config_files": {
    "/etc/angie/angie.conf": "...",
    "/etc/angie/mime.types": "..."
  }
}
```

默认情况下, 该对象处于禁用状态, 因为配置文件可能包含额外的敏感或机密信息。

## 指标

Angie 在 API 的 `/status/` 部分中公开使用指标; 可以通过定义相应的 `location` 来访问它。完全访问:

```
location /status/ {
    api /status/;
}
```

子树访问, 之前已经讨论过:

```
location /stats/ {
    api /status/http/server_zones/;
}
```

## 示例配置

配置包括 `location /status/`、`resolver`、`http upstream`、`http server`、`location`、`cache`、`limit_conn` (在 `http` 中) 和 `limit_req` 区域:

```
http {

    resolver 127.0.0.53 status_zone=resolver_zone;
    proxy_cache_path /var/cache/angie/cache keys_zone=cache_zone:2m;
    limit_conn_zone $binary_remote_addr zone=limit_conn_zone:10m;
    limit_req_zone $binary_remote_addr zone=limit_req_zone:10m rate=1r/s;

    upstream upstream {
        zone upstream 256k;
        server backend.example.com service=_example._tcp resolve max_conns=5;
        keepalive 4;
    }

    server {
        server_name www.example.com;
        listen 443 ssl;

        status_zone http_server_zone;
        proxy_cache cache_zone;

        access_log /var/log/access.log main;

        location / {
            root /usr/share/angie/html;
            status_zone location_zone;
            limit_conn limit_conn_zone 1;
            limit_req zone=limit_req_zone burst=5;
        }
    }
}
```



```
location /status/ {
    api /status/;

    allow 127.0.0.1;
    deny all;
}

}
```

响应 curl https://www.example.com/status/ 时返回以下 JSON:

### JSON 树

```
{
  "angie": {
    "version": "1.8.0",
    "address": "192.168.16.5",
    "generation": 1,
    "load_time": "2024-12-19T12:58:39.789Z"
  },
  "connections": {
    "accepted": 2257,
    "dropped": 0,
    "active": 3,
    "idle": 1
  },
  "slabs": {
    "cache_zone": {
      "pages": {
        "used": 2,
        "free": 506
      }
    },
    "slots": {
      "64": {
        "used": 1,
        "free": 63,
        "reqs": 1,
        "fails": 0
      },
      "512": {
        "used": 1,
        "free": 7,

```

```
        "reqs":1,
        "fails":0
    }
}
},

"limit_conn_zone": {
    "pages": {
        "used":2,
        "free":2542
    },

    "slots": {
        "64": {
            "used":1,
            "free":63,
            "reqs":74,
            "fails":0
        },

        "128": {
            "used":1,
            "free":31,
            "reqs":1,
            "fails":0
        }
    }
},

"limit_req_zone": {
    "pages": {
        "used":2,
        "free":2542
    },

    "slots": {
        "64": {
            "used":1,
            "free":63,
            "reqs":1,
            "fails":0
        },

        "128": {
            "used":2,
            "free":30,
            "reqs":3,
            "fails":0
        }
    }
}
```

```
    }
  }
},
"http": {
  "server_zones": {
    "http_server_zone": {
      "ssl": {
        "handshaked":4174,
        "reuses":0,
        "timedout":0,
        "failed":0
      },
      "requests": {
        "total":4327,
        "processing":0,
        "discarded":8
      },
      "responses": {
        "200":4305,
        "302":12,
        "404":4
      },
      "data": {
        "received":733955,
        "sent":59207757
      }
    }
  },
  "location_zones": {
    "location_zone": {
      "requests": {
        "total":4158,
        "discarded":0
      },
      "responses": {
        "200":4157,
        "304":1
      },
      "data": {
        "received":538200,
```

```
        "sent":177606236
      }
    }
  },
  "caches": {
    "cache_zone": {
      "size":0,
      "cold":false,
      "hit": {
        "responses":0,
        "bytes":0
      },
      "stale": {
        "responses":0,
        "bytes":0
      },
      "updating": {
        "responses":0,
        "bytes":0
      },
      "revalidated": {
        "responses":0,
        "bytes":0
      },
      "miss": {
        "responses":0,
        "bytes":0,
        "responses_written":0,
        "bytes_written":0
      },
      "expired": {
        "responses":0,
        "bytes":0,
        "responses_written":0,
        "bytes_written":0
      },
      "bypass": {
        "responses":0,
        "bytes":0,
        "responses_written":0,
        "bytes_written":0
      }
    }
  }
}
```

```
    }
  },

  "limit_conns": {
    "limit_conn_zone": {
      "passed":73,
      "skipped":0,
      "rejected":0,
      "exhausted":0
    }
  },

  "limit_reqs": {
    "limit_req_zone": {
      "passed":54816,
      "skipped":0,
      "delayed":65,
      "rejected":26,
      "exhausted":0
    }
  },

  "upstreams": {
    "upstream": {
      "peers": {
        "192.168.16.4:80": {
          "server":"backend.example.com",
          "service":"_example._tcp",
          "backup":false,
          "weight":5,
          "state":"up",
          "selected": {
            "current":2,
            "total":232
          },
          "max_conns":5,
          "responses": {
            "200":222,
            "302":12
          },
          "data": {
            "sent":543866,
            "received":27349934
          },
          "health": {
```

```
        "fails":0,
        "unavailable":0,
        "downtime":0
    },

    "sid":"<server_id>"
}
},

"keepalive":2
}
},

"resolvers": {
    "resolver_zone": {
        "queries": {
            "name":442,
            "srv":2,
            "addr":0
        },

        "responses": {
            "success":440,
            "timedout":1,
            "format_error":0,
            "server_failure":1,
            "not_found":1,
            "unimplemented":0,
            "refused":1,
            "other":0
        }
    }
}
}
```

每个 JSON 分支都可以单独请求, 只需根据请求进行相应构建, 例如:

```
$ curl https://www.example.com/status/angle
$ curl https://www.example.com/status/connections
$ curl https://www.example.com/status/slabs
$ curl https://www.example.com/status/slabs/<zone>/slots
$ curl https://www.example.com/status/slabs/<zone>/slots/64
$ curl https://www.example.com/status/http/
$ curl https://www.example.com/status/http/server_zones
$ curl https://www.example.com/status/http/server_zones/<http_server_zone>
$ curl https://www.example.com/status/http/server_zones/<http_server_zone>/ssl
```

**i** 备注

默认情况下, 该模块使用 ISO 8601 字符串表示日期值; 要改用整数的 epoch 格式, 可以在查询字符串中添加 `date=epoch` 参数:

```
$ curl https://www.example.com/status/angie/load_time

"2024-04-01T00:59:59+01:00"

$ curl https://www.example.com/status/angie/load_time?date=epoch

1711929599
```

## 服务器状态

`/status/angie`

```
{
  "version": "1.8.0",
  "address": "192.168.16.5",
  "generation": 1,
  "load_time": "2024-12-19T16:15:43.805Z"
  "config_files": {
    "/etc/angie/angie.conf": "...",
    "/etc/angie/mime.types": "..."
  }
}
```

<code>version</code>	字符串; 当前运行的 Angie Web 服务器版本
<code>build</code>	字符串; 在编译时指定的特定构建名称
<code>address</code>	字符串; 接受 API 请求的服务器地址
<code>generation</code>	数字; 自上次启动以来配置重载的总次数
<code>load_time</code>	字符串或数字; 上次配置重载的时间, 格式为日期; 字符串具有毫秒分辨率
<code>config_files</code>	对象; 其成员是当前服务器实例加载的所有 Angie 配置文件的绝对路径, 它们的值是文件内容的字符串表示, 例如:

```
{
  "/etc/angie/angie.conf": "server {\n  listen 80;\n  # ... \n\n}"
}
```

**小心**

仅当启用了 `api_config_files` 指令时, `config_files` 对象才会出现在 `/status/angie/` 中。

## 连接全局指标

`/status/connections`

```
{
  "accepted": 2257,
  "dropped": 0,
  "active": 3,
  "idle": 1
}
```

accepted	数字; 已接受的客户端连接总数
dropped	数字; 已丢弃的客户端连接总数
active	数字; 当前活跃的客户端连接数
idle	数字; 当前空闲的客户端连接数

## 共享内存区域的 Slab 分配器指标

`/status/slabs/<zone>`

提供使用 slab 分配的共享内存区域的使用统计信息, 例如 `limit_conn`、`limit_req` 和 `HTTP 缓存`:

```
limit_conn_zone $binary_remote_addr zone=limit_conn_zone:10m;
limit_req_zone $binary_remote_addr zone=limit_req_zone:10m rate=1r/s;
proxy_cache cache_zone;
```

指定的共享内存区域将收集以下统计信息:

pages	对象; 内存页面统计信息
used	数字; 当前使用的内存页面数量
free	数字; 当前空闲的内存页面数量
slots	对象; 每个槽大小的内存槽统计信息。slots 对象包含请求的内存槽大小的字段 (例如 8、16、32 等, 最大为页面大小的一半)
used	数字; 当前指定大小的已使用内存槽数量
free	数字; 当前指定大小的空闲内存槽数量
reqs	数字; 尝试分配指定大小内存槽的总次数
fails	数字; 尝试分配指定大小内存槽失败的次数

示例:

```
{
  "pages": {
    "used": 2,
```



```

    "free": 506
  },

  "slots": {
    "64": {
      "used": 1,
      "free": 63,
      "reqs": 1,
      "fails": 0
    }
  }
}

```

### 解析器 DNS 查询

`/status/resolvers/<zone>`

要收集解析器统计信息, 必须在 `resolver` 指令中设置 `status_zone` 参数 (参见 *HTTP* 和 *Stream*):

```
resolver 127.0.0.53 status_zone=resolver_zone;
```

指定的共享内存区域将收集以下统计信息:

<code>queries</code>	对象; 查询统计信息
<code>name</code>	数字; 将名称解析为地址的查询次数 (A 和 AAAA 查询)
<code>srv</code>	数字; 将服务解析为地址的查询次数 (SRV 查询)
<code>addr</code>	数字; 将地址解析为名称的查询次数 (PTR 查询)
<code>responses</code>	对象; 响应统计信息
<code>success</code>	数字; 成功的响应次数
<code>timeout</code>	数字; 查询超时的次数
<code>format_error</code>	数字; 响应码为 1 (格式错误) 的响应次数
<code>server_failure</code>	数字; 响应码为 2 (服务器故障) 的响应次数
<code>not_found</code>	数字; 响应码为 3 (名称错误) 的响应次数
<code>unimplemented</code>	数字; 响应码为 4 (未实现) 的响应次数
<code>refused</code>	数字; 响应码为 5 (被拒绝) 的响应次数
<code>other</code>	数字; 使用其他非零响应码完成的查询次数
<code>sent</code>	对象; 已发送的 DNS 查询统计信息
<code>a</code>	数字; A 类型查询次数
<code>aaaa</code>	数字; AAAA 类型查询次数
<code>ptr</code>	数字; PTR 类型查询次数
<code>srv</code>	数字; SRV 类型查询次数

响应代码的描述详见 RFC 1035, 第 4.1.1 节。

各种 DNS 记录类型的详细信息参见 RFC 1035、RFC 2782 和 RFC 3596。

示例:

```
{
  "queries": {
    "name": 442,
    "srv": 2,
    "addr": 0
  },
  "responses": {
    "success": 440,
    "timedout": 1,
    "format_error": 0,
    "server_failure": 1,
    "not_found": 1,
    "unimplemented": 0,
    "refused": 1,
    "other": 0
  },
  "sent": {
    "a": 185,
    "aaaa": 245,
    "srv": 2,
    "ptr": 12
  }
}
```

## HTTP 服务器和位置

`/status/http/server_zones/<zone>`

要收集 `server` 指标, 请在 `server` 上下文中设置 `status_zone` 指令:

```
server {
  ...
  status_zone server_zone;
}
```

指定的共享内存区域将收集以下统计信息:

ssl	对象; SSL 统计信息。仅在 server 设置了 listen ssl; 时显示
handshaked	数字; 成功的 SSL 握手总次数
reuses	数字; SSL 握手期间会话重用的总次数
timedout	数字; SSL 握手超时的总次数
failed	数字; SSL 握手失败的总次数
requests	对象; 请求统计信息
total	数字; 客户端请求的总次数
processing	数字; 当前正在处理的客户端请求数量
discarded	数字; 未发送响应而完成的客户端请求总数
responses	对象; 响应统计信息
<code>	数字; 具有状态码 <code> (100-599) 的非零响应次数
xxx	数字; 具有其他状态码的非零响应次数
data	对象; 数据统计信息
received	数字; 从客户端接收的字节总数
sent	数字; 发送到客户端的字节总数

示例:

```
"ssl": {
  "handshaked": 4174,
  "reuses": 0,
  "timedout": 0,
  "failed": 0
},

"requests": {
  "total": 4327,
  "processing": 0,
  "discarded": 0
},

"responses": {
  "200": 4305,
  "302": 6,
  "304": 12,
  "404": 4
},

"data": {
  "received": 733955,
  "sent": 59207757
}
```

```
/status/http/location_zones/<zone>
```

要收集 location 指标, 请在 *location* 或 *if in location* 的上下文中设置 *status\_zone* 指令:

```
location / {
    root /usr/share/angie/html;
    status_zone location_zone;

    if ($request_uri ~* "~/condition") {
        # ...
        status_zone if_location_zone;
    }
}
```

指定的共享内存区域将收集以下统计信息:

requests	对象; 请求统计信息
total	数字; 客户端请求的总次数
discarded	数字; 未发送响应而完成的客户端请求总数
responses	对象; 响应统计信息
<code>	数字; 具有状态码 <code> (100-599) 的非零响应次数
xxx	数字; 具有其他状态码的非零响应次数
data	对象; 数据统计信息
received	数字; 从客户端接收的字节总数
sent	数字; 发送到客户端的字节总数

示例:

```
{
  "requests": {
    "total": 4158,
    "discarded": 0
  },
  "responses": {
    "200": 4157,
    "304": 1
  },
  "data": {
    "received": 538200,
    "sent": 177606236
  }
}
```

## 流服务器

`/status/stream/server_zones/<zone>`

要收集 `server` 指标, 请在 `server` 上下文中设置 `status_zone` 指令:

```
server {
    ...
    status_zone server_zone;
}
```

指定的共享内存区域将收集以下统计信息:

<code>ssl</code>	对象; SSL 统计信息。仅在 <code>server</code> 设置了 <code>listen ssl;</code> 时显示
<code>handshaked</code>	数字; 成功的 SSL 握手总次数
<code>reuses</code>	数字; SSL 握手期间会话重用的总次数
<code>timedout</code>	数字; SSL 握手超时的总次数
<code>failed</code>	数字; SSL 握手失败的总次数
<code>connections</code>	对象; 连接统计信息
<code>total</code>	数字; 客户端连接的总次数
<code>processing</code>	数字; 当前正在处理的客户端连接数
<code>discarded</code>	数字; 未创建会话而完成的客户端连接总数
<code>relayed</code>	数字; 使用 <code>pass</code> 指令转发到其他监听端口的客户端连接总数
<code>sessions</code>	对象; 会话统计信息
<code>success</code>	数字; 以代码 200 成功完成的会话数
<code>invalid</code>	数字; 以代码 400 完成的会话数, 表示无法解析客户端数据 (例如 PROXY 协议头部)
<code>forbidden</code>	数字; 以代码 403 完成的会话数, 表示访问被拒绝 (例如, 限制了某些客户端地址的访问)
<code>internal_error</code>	数字; 以代码 500 完成的会话数, 表示服务器内部错误
<code>bad_gateway</code>	数字; 以代码 502 完成的会话数, 表示网关错误 (例如, 上游服务器不可选或不可达)
<code>service_unavailable</code>	数字; 以代码 503 完成的会话数, 表示服务不可用 (例如, 限制了连接数量)
<code>data</code>	对象; 数据统计信息
<code>received</code>	数字; 从客户端接收的字节总数
<code>sent</code>	数字; 发送到客户端的字节总数

示例:

```
{
  "ssl": {
    "handshaked": 24,
    "reuses": 0,
    "timedout": 0,
    "failed": 0
  }
}
```

```
},  
  
"connections": {  
  "total": 24,  
  "processing": 1,  
  "discarded": 0,  
  "passed": 2  
},  
  
"sessions": {  
  "success": 24,  
  "invalid": 0,  
  "forbidden": 0,  
  "internal_error": 0,  
  "bad_gateway": 0,  
  "service_unavailable": 0  
},  
  
"data": {  
  "received": 2762947,  
  "sent": 53495723  
}  
}
```

## HTTP 缓存

```
proxy_cache cache_zone;
```

```
/status/http/caches/<cache>
```

对于使用 `proxy_cache` 配置的每个区域, 将存储以下数据:

```
{  
  "name_zone": {  
    "size": 0,  
    "cold": false,  
    "hit": {  
      "responses": 0,  
      "bytes": 0  
    },  
  
    "stale": {  
      "responses": 0,  
      "bytes": 0  
    },  
  },  
}
```

```
"updating": {
  "responses": 0,
  "bytes": 0
},

"revalidated": {
  "responses": 0,
  "bytes": 0
},

"miss": {
  "responses": 0,
  "bytes": 0,
  "responses_written": 0,
  "bytes_written": 0
},

"expired": {
  "responses": 0,
  "bytes": 0,
  "responses_written": 0,
  "bytes_written": 0
},

"bypass": {
  "responses": 0,
  "bytes": 0,
  "responses_written": 0,
  "bytes_written": 0
}
}
```

size	数字; 当前缓存的大小
max_size	数字; 缓存的最大配置限制
cold	布尔值; 在 缓存加载器从磁盘加载数据时为 true
hit	对象; 有效缓存响应的统计信息 ( <i>proxy_cache_valid</i> )
responses	数字; 从缓存读取的响应总数
bytes	数字; 从缓存读取的字节总数
stale	对象; 从缓存中读取的过期响应的统计信息 ( <i>proxy_cache_use_stale</i> )
responses	数字; 从缓存读取的响应总数
bytes	数字; 从缓存读取的字节总数
updating	对象; 在 响应更新时从缓存中读取的过期响应的统计信息 ( <i>proxy_cache_use_stale updating</i> )
responses	数字; 从缓存读取的响应总数
bytes	数字; 从缓存读取的字节总数
revalidated	对象; 从缓存中读取的过期并重新验证的响应统计信息 ( <i>proxy_cache_revalidate</i> )
responses	数字; 从缓存读取的响应总数
bytes	数字; 从缓存读取的字节总数
miss	对象; 在缓存中未找到的响应统计信息
responses	数字; 相应的响应总数
bytes	数字; 从代理服务器读取的字节总数
responses_written	数字; 写入缓存的响应总数
bytes_written	数字; 写入缓存的字节总数
expired	对象; 未从缓存中取出的过期响应统计信息
responses	数字; 相应的响应总数
bytes	数字; 从代理服务器读取的字节总数
responses_written	数字; 写入缓存的响应总数
bytes_written	数字; 写入缓存的字节总数
bypass	对象; 未在缓存中查找的响应统计信息 ( <i>proxy_cache_bypass</i> )
responses	数字; 相应的响应总数
bytes	数字; 从代理服务器读取的字节总数
responses_written	数字; 写入缓存的响应总数
bytes_written	数字; 写入缓存的字节总数

Added in version 1.2.0: PRO

在 Angie PRO 中, 如果使用 *proxy\_cache\_path* 指令启用了 缓存分片, 则各个分片将作为 shards 对象的成员公开:

shards	对象; 将各个分片列为成员
<shard>	对象; 表示一个分片, 其名称为缓存路径
sizes	数字; 分片的当前大小
max_size	数字; 分片的最大大小 (如果已配置)
cold	布尔值; 在 缓存加载器从磁盘加载数据时为 true



```
{
  "name_zone": {
    "shards": {
      "/path/to/shard1": {
        "size": 0,
        "cold": false
      },

      "/path/to/shard2": {
        "size": 0,
        "cold": false
      }
    }
  }
}
```

### limit\_conn

```
limit_conn_zone $binary_remote_addr zone=limit_conn_zone:10m;
```

`/status/http/limit_conns/<zone>`, `/status/stream/limit_conns/<zone>`

为每个配置在 `http` 中的 `limit_conn` 或 `stream` 中的 `limit_conn` 上下文提供以下字段的对象:

```
{
  "passed": 73,
  "skipped": 0,
  "rejected": 0,
  "exhausted": 0
}
```

<code>passed</code>	数字; 通过的连接总数
<code>skipped</code>	数字; 因键长度为零或超过 255 字节而通过的连接总数
<code>rejected</code>	数字; 超过配置限制而被拒绝的连接总数
<code>exhausted</code>	数字; 由于区域存储耗尽而被拒绝的连接总数

## limit\_req

```
limit_req_zone $binary_remote_addr zone=limit_req_zone:10m rate=1r/s;
```

/status/http/limit\_reqs/<zone>

为每个配置的`limit_req` 提供以下字段的对象:

```
{
  "passed": 54816,
  "skipped": 0,
  "delayed": 65,
  "rejected": 26,
  "exhausted": 0
}
```

passed	数字; 通过的请求总数
skipped	数字; 因键长度为零或超过 65535 字节而通过的请求总数
delayed	数字; 被延迟的请求总数
rejected	数字; 被拒绝的请求总数
exhausted	数字; 由于区域存储耗尽而被拒绝的请求总数

## HTTP 上游

Added in version 1.1.0.

要启用以下指标的收集, 请在`upstream` 上下文中设置`zone` 指令, 例如:

```
upstream upstream {
  zone upstream 256k;
  server backend.example.com service=_example._tcp resolve max_conns=5;
  keepalive 4;
}
```

/status/http/upstreams/<upstream>

其中 `<upstream>` 是使用`zone` 指令指定的任何`upstream` 的名称:

```
{
  "peers": {
    "192.168.16.4:80": {
      "server": "backend.example.com",
      "service": "_example._tcp",

```

```
    "backup": false,  
    "weight": 5,  
    "state": "up",  
    "selected": {  
      "current": 2,  
      "total": 232  
    },  
  
    "max_conns": 5,  
    "responses": {  
      "200": 222,  
      "302": 12  
    },  
  
    "data": {  
      "sent": 543866,  
      "received": 27349934  
    },  
  
    "health": {  
      "fails": 0,  
      "unavailable": 0,  
      "downtime": 0  
    },  
  
    "sid": "<server_id>"  
  }  
},  
  
"keepalive": 2  
}
```

<code>peers</code>	对象; 包含上游服务器的各个节点指标, 每个子对象的名称是节点地址的规范表示。每个子对象的成员:
<code>server</code>	字符串; <code>server</code> 指令的参数
<code>service</code>	字符串; 在 <code>server</code> 指令中指定的服务名称 (如果已配置)
<code>slow_start</code> (PRO 1.4.0+)	数字; 服务器的 <code>slow_start</code> 指定值, 单位为秒。 在动态配置 API 的相应子部分中设置此值时, 可以指定数字或具有毫秒精度的时间值。
<code>backup</code>	布尔值; 对于备份服务器为 <code>true</code>
<code>weight</code>	数字; 配置的权重
<code>state</code>	字符串; 当前节点状态: <ul style="list-style-type: none"> <li>• <code>checking</code> (PRO): 设置为 <code>essential</code>, 当前正在检查, 仅发送探测请求</li> <li>• <code>down</code>: 手动禁用, 不发送请求</li> <li>• <code>draining</code> (PRO): 类似于 <code>down</code>, 但通过 <code>sticky</code> 绑定的会话请求仍然会被发送</li> <li>• <code>recovering</code>: 故障恢复中, 遵循 <code>slow_start</code>, 逐步增加请求数</li> <li>• <code>unavailable</code>: 达到 <code>max_fails</code> 限制, 在 <code>fail_timeout</code> 间隔重试</li> <li>• <code>unhealthy</code> (PRO): 节点未正常运行, 仅发送探测请求</li> <li>• <code>up</code>: 正常运行, 请求正常发送</li> </ul>
<code>selected</code>	对象; 节点选择统计信息
<code>current</code>	数字; 当前与节点的连接数
<code>total</code>	数字; 转发到节点的请求总数
<code>last</code>	字符串或数字; 上次选择节点的时间, 格式为日期
<code>max_conns</code>	数字; 配置的同时连接数最大限制 (如果指定)
<code>responses</code>	对象; 响应统计信息
<code>&lt;code&gt;</code>	数字; 具有状态码 <code>&lt;code&gt;</code> (100-599) 的非零响应次数
<code>xxx</code>	数字; 其他状态码的非零响应次数
<code>data</code>	对象; 数据统计信息
<code>received</code>	数字; 从节点接收的字节总数
<code>sent</code>	数字; 发送到节点的字节总数
<code>health</code>	对象; 健康统计信息
<code>fails</code>	数字; 与节点通信失败的总尝试次数
<code>unavailable</code>	数字; 节点因达到 <code>max_fails</code> 限制而变为 <code>unavailable</code> 的次数
<code>downtime</code>	数字; 节点因 <code>unavailable</code> 无法选择的总时间 (单位: 毫秒)
<code>downstart</code>	字符串或数字; 节点变为 <code>unavailable</code> 的时间, 格式为日期
<code>header_time</code> (PRO 1.3.0+)	数字; 从节点接收响应头的平均时间 (单位: 毫秒); 参见 <code>response_time_factor</code> (PRO)
<code>response_time</code> (PRO 1.3.0+)	数字; 接收完整节点响应的平均时间 (单位: 毫秒); 参见 <code>response_time_factor</code> (PRO)
<code>sid</code>	字符串; 上游组中服务器的配置 ID
<code>keepalive</code>	数字; 当前缓存的连接数

## health/probes (PRO)

在 1.2.0 版本发生变更: PRO

如果上游配置了 *upstream\_probe (PRO)* 探测, health 对象还将包含一个 probes 子对象, 用于存储节点的健康探测计数器, 同时节点的 state 也可以是 checking 和 unhealthy, 除了上表列出的值之外:

```
{
  "192.168.16.4:80": {
    "state": "unhealthy",
    "...": "...",
    "health": {
      "...": "...",
      "probes": {
        "count": 10,
        "fails": 10,
        "last": "2024-12-19T09:56:07Z"
      }
    }
  }
}
```

state 的 checking 值不会计为 downtime, 表示配置为 essential 的探测节点尚未被检查; unhealthy 值表示节点运行异常。这两个状态都意味着该节点不会参与负载均衡。有关健康探测的详细信息, 请参阅上游探测。

probes 中的计数器:

count	数字; 此节点的总探测次数
fails	数字; 失败的探测总次数
last	字符串或数字; 上次探测时间, 格式为日期

## queue

在 1.4.0 版本发生变更.

如果为上游配置了请求队列, 上游对象还将包含一个嵌套的 queue 对象, 用于存储队列中的请求计数器:

```
{
  "queue": {
    "queued": 20112,
    "waiting": 1011,
    "dropped": 6031,
    "timedout": 560,
    "overflows": 13
  }
}
```

计数器值会汇总所有工作进程的数据:

queued	数字; 进入队列的请求总数
waiting	数字; 当前在队列中的请求数量
dropped	数字; 由于客户端提前关闭连接而从队列中移除的请求总数
timedout	数字; 由于超时而从队列中移除的请求总数
overflows	数字; 队列溢出发生的总次数

## 流上游

要启用以下指标的收集, 请在`upstream`上下文中设置`zone`指令, 例如:

```
upstream upstream {
    zone upstream 256k;
    server backend.example.com service=_example._tcp resolve max_conns=5;
    keepalive 4;
}
```

`/status/stream/upstreams/<upstream>`

这里, `<upstream>` 是通过`zone`指令配置的上游的名称。

```
{
  "peers": {
    "192.168.16.4:1935": {
      "server": "backend.example.com",
      "service": "_example._tcp",
      "backup": false,
      "weight": 5,
      "state": "up",
      "selected": {
        "current": 2,
        "total": 232
      },
      "max_conns": 5,
      "data": {
        "sent": 543866,
        "received": 27349934
      },
      "health": {
        "fails": 0,
        "unavailable": 0,
        "downtime": 0
      }
    }
  }
}
```

```

    }
  }
}

```

<b>peers</b>	对象; 包含上游服务器节点的各项指标, 每个子对象的名称是节点地址的规范表示。每个子对象的成员:
<b>server</b>	字符串; 由 <i>server</i> 指令设置的地址
<b>service</b>	字符串; 服务名称, 如果在 <i>server</i> 指令中设置
<b>slow_start</b> (PRO 1.4.0+)	数字; 服务器的 <i>slow_start</i> 值, 单位为秒。 在动态配置 API 的相应子部分中, 可指定数字或具有毫秒精度的时间。
<b>backup</b>	布尔值; 备份服务器时为 <code>true</code>
<b>weight</b>	数字; 节点的权重
<b>state</b>	字符串; 节点的当前状态: <ul style="list-style-type: none"> <li>• <code>up</code>: 正常运行, 请求正常发送</li> <li>• <code>down</code>: 手动禁用, 不发送请求</li> <li>• <code>draining</code> (PRO): 类似于 <code>down</code>, 但通过 <i>sticky</i> 绑定的会话请求仍然发送</li> <li>• <code>unavailable</code>: 达到 <i>max_fails</i> 限制, 在 <i>fail_timeout</i> 间隔尝试重试</li> <li>• <code>recovering</code>: 故障恢复中, 遵循 <i>slow_start</i>, 逐步增加请求数</li> <li>• <code>checking</code> (PRO): 设置为 <code>essential</code>, 当前正在检查, 仅发送探测请求</li> <li>• <code>unhealthy</code> (PRO): 节点运行异常, 仅发送探测请求</li> </ul>
<b>selected</b>	对象; 节点的选择统计信息
<b>current</b>	数字; 当前与节点的连接数
<b>total</b>	数字; 转发到节点的连接总数
<b>last</b>	字符串或数字; 上次选择该节点的时间, 格式为日期
<b>max_conns</b>	数字; 节点的最大同时连接数 (如已设置)
<b>data</b>	对象; 数据传输统计信息
<b>received</b>	数字; 从节点接收的总字节数
<b>sent</b>	数字; 发送到节点的总字节数
<b>health</b>	对象; 节点健康统计信息
<b>fails</b>	数字; 与节点通信失败的总次数
<b>unavailable</b>	数字; 节点因达到 <i>max_fails</i> 而变为 <code>unavailable</code> 的次数
<b>downtime</b>	数字; 节点因 <code>unavailable</code> 无法选择的总时间 (单位: 毫秒)
<b>downstart</b>	字符串或数字; 节点上次变为 <code>unavailable</code> 的时间, 格式为日期
<b>connect_time</b> (PRO 1.4.0+)	数字; 与节点建立连接的平均时间 (单位: 毫秒); 参见 <i>response_time_factor</i> (PRO) 指令。
<b>first_byte_time</b> (PRO 1.4.0+)	数字; 接收来自节点的第一个字节的平均时间 (单位: 毫秒) ; 参见 <i>response_time_factor</i> (PRO) 指令。
<b>last_byte_time</b> (PRO 1.4.0+)	数字; 接收来自节点的完整响应的平均时间 (单位: 毫秒) ; 参见 <i>response_time_factor</i> (PRO) 指令。

在 1.4.0 版本发生变更: PRO

在 Angie PRO 中, 如果上游配置了 *upstream\_probe* (PRO) 探测, `health` 对象还将包含一个 `probes` 子对象, 用于存储节点的健康探测计数器, 同时节点的 `state` 也可以是 `checking` 和 `unhealthy`, 除了上述状态之外:

```
{
  "192.168.16.4:80": {
    "state": "unhealthy",
    "...": "...",
    "health": {
      "...": "...",
      "probes": {
        "count": 2,
        "fails": 2,
        "last": "2024-12-19T11:03:54Z"
      }
    }
  }
}
```

`state` 的 `checking` 值不会计为 `downtime`, 表示配置为 `essential` 的探测节点尚未被检查; `unhealthy` 值表示节点运行异常。这两个状态都意味着节点不会参与负载均衡。有关健康探测的详细信息, 请参阅[上游探测](#)。

`probes` 中的计数器:

<code>count</code>	数字; 此节点的总探测次数
<code>fails</code>	数字; 失败的探测总次数
<code>last</code>	字符串或数字; 上次探测时间, 格式为日期

### 动态配置 API (仅限 PRO)

Added in version 1.2.0.

API 包括一个 `/config` 部分, 允许通过 JSON 进行动态更新 Angie 配置, 支持 `PUT`、`PATCH` 和 `DELETE` HTTP 请求。所有更新都是原子性的; 新设置要么整体应用, 要么完全不应用。发生错误时, Angie 会报告原因。



## /config 的子部分

目前, HTTP 和 Stream 模块中的上游服务器配置可通过 /config 进行动态更新, 参见 *HTTP* 和 *Stream* 部分; 支持动态配置的设置数量正在不断增加。

/config/http/upstreams/<upstream>/servers/<name>

用于配置单个上游节点, 包括删除现有节点或添加新节点。

URI 路径参数:

<upstream>	上游名称; 要通过 /config 进行配置, 必须设置 <i>zone</i> 指令, 定义一个共享内存区域。
<name>	上游中的节点名称, 格式为 <service>@<host>, 其中: <ul style="list-style-type: none"><li>• &lt;service&gt;@ 是一个可选的服务名称, 用于 SRV 记录解析。</li><li>• &lt;host&gt; 是服务的域名 (如果存在 <i>resolve</i> 指令) 或其 IP; 可选端口也可定义。</li></ul>

例如, 以下配置:

```
upstream backend {  
    server backend.example.com:8080 service=_http._tcp resolve;  
    server 127.0.0.1;  
    zone backend 1m;  
}
```

允许以下节点名称:

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/_http._tcp@backend.example.  
↪ com:8080/  
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/127.0.0.1/
```

此 API 子部分允许设置 *weight*、*max\_conns*、*max\_fails*、*fail\_timeout*、*backup*、*down* 和 *sid* 参数, 具体说明见 *server*。

### **i** 备注

这里没有单独的 *drain* 选项; 要启用 *drain*, 请将 *down* 设置为字符串值 *drain*:

```
$ curl -X PUT -d \"drain\" \  
http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/down
```

示例:

```
curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com?defaults=on
```

```
{
  "weight": 1,
  "max_conns": 0,
  "max_fails": 1,
  "fail_timeout": 10,
  "backup": true,
  "down": false,
  "sid": ""
}
```

实际可用的参数受当前上游的负载均衡方法限制。例如, 如果上游配置为 `random`:

```
upstream backend {
  zone backend 256k;
  server backend.example.com resolve max_conns=5;
  random;
}
```

将无法添加定义 `backup` 的新节点:

```
$ curl -X PUT -d '{ "backup": true }' \
  http://127.0.0.1/config/http/upstreams/backend/servers/backend1.example.com
```

```
{
  "error": "FormatError",
  "description": "The \"backup\" field is unknown."
}
```

#### 备注

即使使用兼容的负载均衡方法, `backup` 参数也只能在新节点创建时设置。

删除服务器时, 可以使用 `connection_drop=<value>` 参数 (PRO), 以覆盖 `proxy_connection_drop`、`grpc_connection_drop`、`fastcgi_connection_drop`、`scgi_connection_drop` 和 `uwsgi_connection_drop` 的设置:

```
$ curl -X DELETE \
  http://127.0.0.1/config/http/upstreams/backend/servers/backend1.example.com?connection_
↪drop=off

$ curl -X DELETE \
  http://127.0.0.1/config/http/upstreams/backend/servers/backend2.example.com?connection_
↪drop=on
```

```
$ curl -X DELETE \  
  http://127.0.0.1/config/http/upstreams/backend/servers/backend3.example.com?connection_  
↪drop=1000
```

/config/stream/upstreams/<upstream>/servers/<name>

启用配置单个上游节点, 包括删除现有节点或添加新节点。

URI 路径参数:

<upstream>	上游的名称; 要通过 /config 进行配置, 必须配置 <i>zone</i> 指令, 定义一个共享内存区域。
<name>	上游中的节点名称, 格式为 <service>@<host>, 其中: <ul style="list-style-type: none"><li>• &lt;service&gt;@ 是可选的服务名称, 用于 SRV 记录解析。</li><li>• &lt;host&gt; 是服务的域名 (如果存在 <i>resolve</i>) 或其 IP; 可选端口也可定义。</li></ul>

例如, 以下配置:

```
upstream backend {  
  server backend.example.com:8080 service=_example._tcp resolve;  
  server 127.0.0.1:12345;  
  zone backend 1m;  
}
```

允许以下节点名称:

```
$ curl http://127.0.0.1/config/stream/upstreams/backend/servers/_example._tcp@backend.example.  
↪com:8080/  
$ curl http://127.0.0.1/config/stream/upstreams/backend/servers/127.0.0.1:12345/
```

此 API 子部分允许设置 *weight*、*max\_conns*、*max\_fails*、*fail\_timeout*、*backup* 和 *down* 参数, 具体说明见 *server*。

#### 备注

这里没有单独的 *drain* 选项; 要启用 *drain*, 请将 *down* 设置为字符串值 *drain*:

```
$ curl -X PUT -d \"drain\" \  
  http://127.0.0.1/config/stream/upstreams/backend/servers/backend.example.com/down
```

示例:

```
curl http://127.0.0.1/config/stream/upstreams/backend/servers/backend.example.com?defaults=on
```

```
{
  "weight": 1,
  "max_conns": 0,
  "max_fails": 1,
  "fail_timeout": 10,
  "backup": true,
  "down": false
}
```

实际可用的参数受当前上游的负载均衡方法限制。例如, 如果上游配置为 `random`:

```
upstream backend {
  zone backend 256k;
  server backend.example.com resolve max_conns=5;
  random;
}
```

将无法添加定义 `backup` 的新节点:

```
$ curl -X PUT -d '{"backup": true}' \
  http://127.0.0.1/config/stream/upstreams/backend/servers/backend1.example.com
```

```
{
  "error": "FormatError",
  "description": "The \"backup\" field is unknown."
}
```

#### **i** 备注

即使使用兼容的负载均衡方法, `backup` 参数也只能在新节点创建时设置。

删除服务器时, 可以使用 `connection_drop=<value>` 参数 (PRO), 以覆盖 `proxy_connection_drop` 设置:

```
$ curl -X DELETE \
  http://127.0.0.1/config/stream/upstreams/backend/servers/backend1.example.com?connection_
↵drop=off

$ curl -X DELETE \
  http://127.0.0.1/config/stream/upstreams/backend/servers/backend2.example.com?connection_
↵drop=on

$ curl -X DELETE \
  http://127.0.0.1/config/stream/upstreams/backend/servers/backend3.example.com?connection_
```

```
↔ drop=1000
```

## HTTP 方法

让我们考虑适用于本节的所有 HTTP 方法的语义, 以以下上游配置为例:

```
http {
    # ...

    upstream backend {
        zone upstream 256k;
        server backend.example.com resolve max_conns=5;
        # ...
    }

    server {
        # ...

        location /config/ {
            api /config/;

            allow 127.0.0.1;
            deny all;
        }
    }
}
```

## GET

GET HTTP 方法查询 `/config` 中任何现有路径的实体, 与其他 API 部分类似。

例如, `/config/http/upstreams/backend/servers/` 上游服务器分支支持以下查询:

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/max_conns
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
$ curl http://127.0.0.1/config/http/upstreams/backend/servers
$ # ...
$ curl http://127.0.0.1/config
```

可以使用 `defaults=on` 获取默认参数值:

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers?defaults=on
```

```
{
  "backend.example.com": {
    "weight": 1,
```

```
    "max_conns": 5,  
    "max_fails": 1,  
    "fail_timeout": 10,  
    "backup": false,  
    "down": false,  
    "sid": ""  
  }  
}
```

## PUT

PUT HTTP 方法在指定路径创建一个新的 JSON 实体, 或 完全替换现有实体。

例如, 为 backend 上游中的 backend.example.com 服务器设置未指定的 max\_fails 参数:

```
$ curl -X PUT -d '2' \  
http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/max_fails
```

```
{  
  "success": "Updated",  
  "description": "Existing configuration API entity \"/config/http/upstreams/backend/servers/  
↪backend.example.com/max_fails\" was updated with replacing."  
}
```

验证更改:

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{  
  "max_conns": 5,  
  "max_fails": 2  
}
```

## DELETE

DELETE HTTP 方法删除指定路径下 先前定义的设置, 并恢复默认值 (如果存在)。

例如, 删除 backend 上游中 backend.example.com 服务器的 max\_fails 参数:

```
$ curl -X DELETE \  
http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com/max_fails
```

```
{  
  "success": "Reset",  
  "description": "Configuration API entity \"/config/http/upstreams/backend/servers/backend.
```

```
↔example.com/max_fails\" was reset to default.\"  
}
```

使用 defaults=on 验证更改:

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com?defaults=on
```

```
{  
  "weight": 1,  
  "max_conns": 5,  
  "max_fails": 1,  
  "fail_timeout": 10,  
  "backup": false,  
  "down": false,  
  "sid": ""  
}
```

max\_fails 设置已恢复为默认值。

## PATCH

PATCH HTTP 方法在指定路径创建新实体, 或通过提供 JSON 定义来部分替换或补充现有实体 (参见 RFC 7386)。

该方法的操作方式如下: 如果新定义中的实体存在于配置中, 它们将被覆盖; 否则, 将添加新值。

例如, 更改 backend 上游中 backend.example.com 服务器的 down 设置, 同时保留其他设置不变:

```
$ curl -X PATCH -d '{ "down": true }' \  
  http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{  
  "success": "Updated",  
  "description": "Existing configuration API entity \"/config/http/upstreams/backend/servers/  
↔backend.example.com\" was updated with merging."  
}
```

验证更改:

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{  
  "max_conns": 5,  
  "down": true  
}
```

在 PATCH 请求中提供的 JSON 对象 被合并到现有对象中, 而不像 PUT 那样完全覆盖。

null 值是一个特例; 它用于在合并时 删除特定配置项。

#### 备注

此删除操作与 DELETE 相同; 特别是, 它会恢复默认值。

例如, 同时删除先前添加的 down 设置并更新 max\_conns:

```
$ curl -X PATCH -d '{ "down": null, "max_conns": 6 }' \  
http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{  
  "success": "Updated",  
  "description": "Existing configuration API entity \"/config/http/upstreams/backend/servers/  
↔backend.example.com\" was updated with merging."  
}
```

验证更改:

```
$ curl http://127.0.0.1/config/http/upstreams/backend/servers/backend.example.com
```

```
{  
  "max_conns": 6  
}
```

提供了 null 的 down 参数已被删除, max\_conns 已更新。

## 基础认证

允许通过使用“HTTP 基本认证”协议验证用户名和密码来限制对资源的访问。

访问也可以通过地址 或通过子请求结果 来限制。通过地址和密码同时限制访问由满足 指令控制。

## 配置示例

```
location / {  
  auth_basic "closed site";  
  auth_basic_user_file conf/htpasswd;  
}
```



## 指令

### auth\_basic

语法	<code>auth_basic string   off;</code>
默认	<code>auth_basic off;</code>
上下文	http, server, location, limit_except

通过“HTTP 基本认证”协议启用用户名和密码验证。指定的参数用作 领域。参数值可以包含变量。

off	取消从先前配置级别继承的 <code>auth_basic</code> 指令的效果
-----	--

### auth\_basic\_user\_file

语法	<code>auth_basic_user_file file;</code>
默认	—
上下文	http, server, location, limit_except

指定一个 文件来保存用户名和密码, 格式如下:

```
# 注释 name1:password1 name2:password2:comment name3:password3
```

文件名可以包含变量。

支持以下密码类型:

- 使用 ‘crypt()’ 函数加密; 可以使用 Apache HTTP Server 发行版中的 `htpasswd` 工具或 “`openssl passwd`” 命令生成;
- 使用基于 MD5 的密码算法的 Apache 变体 (apr1) 进行哈希; 可以使用相同的工具生成;
- 按照 RFC 2307 中描述的 “`{scheme}data`” 语法指定; 目前实现的方案包括 ‘PLAIN’ (一个示例, 不应使用)、*SHA* (纯 SHA-1 哈希, 不应使用) 和 ‘SSHA’ (加盐 SHA-1 哈希, 一些软件包使用, 尤其是 OpenLDAP 和 Dovecot)。

#### 小心

SHA 方案的支持仅为从其他 Web 服务器迁移提供帮助。它不应用于新密码, 因为采用的未加盐 SHA-1 哈希易受 ‘彩虹表 <[http://en.wikipedia.org/wiki/Rainbow\\_attack](http://en.wikipedia.org/wiki/Rainbow_attack)>’ 攻击。

## 认证请求

根据子请求的结果实现客户端授权。如果子请求返回 2xx 响应代码, 则允许访问。如果返回 401 或 403, 则拒绝访问, 并返回相应的错误代码。子请求返回的任何其他响应代码都被视为错误。

对于 401 错误, 客户端还会收到来自子请求响应的”WWW-Authenticate”头。

当从源代码构建时, 该模块默认未构建; 应通过 `--with-http_auth_request_module` 构建选项启用。

在来自我们的仓库的包和镜像中, 模块已包含在构建中。

该模块可以与其他访问模块组合, 例如访问控制 和基础认证, 通过 *satisfy* 指令。

## 配置示例

```
location /private/ {
    auth_request /auth;
#    ...
}

location = /auth {
    proxy_pass ...
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
    proxy_set_header X-Original-URI $request_uri;
}
```

## 指令

### auth\_request

语法	<code>auth_request uri   off;</code>
默认值	<code>auth_request off;</code>
上下文	http, server, location

根据子请求的结果启用授权, 并设置子请求将被发送的 URI。

## auth\_request\_set

语法	<code>auth_request_set \$variable value;</code>
默认值	—
上下文	http, server, location

在授权请求完成后, 将请求变量设置为给定值。值可以包含来自授权请求的变量, 例如 `$upstream_http_*`。

## 自动索引

该模块处理以斜杠字符 (/) 结尾的请求, 并生成目录列表。通常, 当 `http_index` 模块找不到索引文件时, 请求会传递给 `http_autoindex` 模块。

## 配置示例

```
location / {  
    autoindex on;  
}
```

## 指令

### autoindex

语法	<code>autoindex on   off;</code>
默认值	<code>autoindex off;</code>
上下文	http, server, location

启用或禁用目录列表输出。

### autoindex\_exact\_size

语法	<code>autoindex_exact_size on   off;</code>
默认值	<code>autoindex_exact_size on;</code>
上下文	http, server, location

对于 HTML 格式, 指定目录列表中是否输出精确的文件大小, 或者将其四舍五入到千字节、兆字节和千兆字节。

## autoindex\_format

语法	<code>autoindex_format html   xml   json   jsonp;</code>
默认值	<code>autoindex_format html;</code>
上下文	<code>http, server, location</code>

设置目录列表的格式。

当使用 JSONP 格式时, 回调函数的名称通过 `callback` 请求参数设置。如果参数缺失或为空值, 则使用 JSON 格式。

XML 输出可以通过 `http_xslt` 模块进行转换。

## autoindex\_localtime

语法	<code>autoindex_localtime on   off;</code>
默认值	<code>autoindex_localtime off;</code>
上下文	<code>http, server, location</code>

对于 HTML 格式, 指定目录列表中的时间是以本地时区还是以 UTC 输出。

## 浏览器

该模块创建的变量值取决于 “User-Agent” 请求头字段的值。

### 变量

#### `$modern_browser`

如果浏览器被识别为现代浏览器, 则等于通过 `modern_browser_value` 指令设置的值;

#### `$ancient_browser`

如果浏览器被识别为旧版浏览器, 则等于通过 `ancient_browser_value` 指令设置的值;

`$msie`

如果浏览器被识别为任何版本的 MSIE, 则等于“1”。

## 配置示例

选择索引文件:

```
modern_browser_value "modern.";

modern_browser msie      5.5;
modern_browser gecko     1.0.0;
modern_browser opera     9.0;
modern_browser safari    413;
modern_browser konqueror 3.0;

index index.${modern_browser}html index.html;
```

旧版浏览器重定向:

```
modern_browser msie      5.0;
modern_browser gecko     0.9.1;
modern_browser opera     8.0;
modern_browser safari    413;
modern_browser konqueror 3.0;

modern_browser unlisted;

ancient_browser Links Lynx netscape4;

if ($ancient_browser) {
    rewrite ^ /ancient.html;
}
```

## 指令

### ancient\_browser

Syntax	<code>ancient_browser <i>string</i> ...;</code>
Default	—
Context	http, server, location

如果“User-Agent”请求头字段中找到任何指定的子字符串, 则浏览器将被视为旧版。特殊字符串“netscape4”对应于正则表达式“^Mozilla/[1-4]”。

### ancient\_browser\_value

Syntax	<code>ancient_browser_value string;</code>
Default	<code>ancient_browser_value 1;</code>
Context	http, server, location

为`$ancient_browser` 变量设置一个值。

### modern\_browser

Syntax	<code>modern_browser browser version;</code> <code>modern_browser unlisted;</code>
Default	—
Context	http, server, location

指定一个版本, 从该版本开始浏览器被视为现代。浏览器可以是以下任何一种: `msie`、`gecko` (基于 Mozilla 的浏览器)、`opera`、`safari` 或 `konqueror`。

版本可以指定为以下格式: `X.X.X.X.X.X` 或 `X.X.X.X`。每种格式的最大值分别为 4000、4000.99、4000.99.99 和 4000.99.99.99。

特殊值 `unlisted` 指定如果浏览器未被 `modern_browser` 和 `ancient_browser` 指令列出, 则视为现代浏览器。否则, 此类浏览器被视为旧版。如果请求未在头中提供“User-Agent”字段, 则浏览器被视为未列出。

### modern\_browser\_value

Syntax	<code>modern_browser_value string;</code>
Default	<code>modern_browser_value 1;</code>
Context	http, server, location

为`$modern_browser` 变量设置一个值。

## 字符集

该模块将指定的字符集添加到“Content-Type”响应头字段。此外, 该模块可以在某些限制条件下将数据从一种字符集转换为另一种字符集:

- 转换是单向的——从服务器到客户端,
- 仅支持单字节字符集的转换
- 或单字节字符集与 UTF-8 之间的转换。

## 配置示例

```
include      conf/koi-win;  
  
charset     windows-1251;  
source_charset koi8-r;
```

## 指令

### charset

语法	<code>charset charset   off;</code>
默认值	<code>charset off;</code>
上下文	http, server, location, if in location

将指定的字符集添加到“Content-Type”响应头字段。如果该字符集与`source_charset`指令中指定的字符集不同, 则会进行转换。

参数 `off` 取消将字符集添加到“Content-Type”响应头字段。

字符集可以用变量定义:

```
charset $charset;
```

在这种情况下, 变量的所有可能值需要至少在配置中以`charset_map`、`charset`或`source_charset`指令的形式出现一次。对于 `utf-8`、`windows-1251` 和 `koi8-r` 字符集, 只需将文件 `conf/koi-win`、`conf/koi-utf` 和 `conf/win-utf` 包含到配置中即可。对于其他字符集, 只需制作一个虚构的转换表, 例如:

```
charset_map iso-8859-5 _ { }
```

此外, 可以在“X-Accel-Charset”响应头字段中设置字符集。可以使用`proxy_ignore_headers`、`fastcgi_ignore_headers`、`uwsgi_ignore_headers`、`scgi_ignore_headers` 和`grpc_ignore_headers` 指令禁用此功能。

## charset\_map

语法	<code>charset_map charset1 charset2 { ... }</code>
默认值	—
上下文	http

描述从一种字符集到另一种字符集的转换表。使用相同的数据构建反向转换表。字符代码以十六进制表示。范围 80-FF 中缺失的字符被替换为“?”。从 UTF-8 转换时, 单字节字符集中缺失的字符被替换为“*?*#XXXX;”。

示例:

```
charset_map koi8-r windows-1251 {
    C0 FE ; # small yu
    C1 E0 ; # small a
    C2 E1 ; # small b
    C3 F6 ; # small ts
}
```

在描述到 UTF-8 的转换表时, UTF-8 字符集的代码应在第二列中给出, 例如:

```
charset_map koi8-r utf-8 {
    C0 D18E ; # small yu
    C1 DOB0 ; # small a
    C2 DOB1 ; # small b
    C3 D186 ; # small ts
}
```

从 koi8-r 到 windows-1251, 以及从 koi8-r 和 windows-1251 到 utf-8 的完整转换表在发行文件 `conf/koi-win`、`conf/koi-utf` 和 `conf/win-utf` 中提供。

## charset\_types

语法	<code>charset_types mime-type ...;</code>
默认值	<code>charset_types text/html text/xml text/plain text/vnd.wap.wml application/javascript application/rss+xml;</code>
上下文	http, server, location

在响应中启用模块处理, 指定的 MIME 类型除了“*text/html*”之外。特殊值“\*”匹配任何 MIME 类型。



## override\_charset

语法	<code>override_charset on   off;</code>
默认值	<code>override_charset off;</code>
上下文	http, server, location, if in location

确定是否应对从代理或 FastCGI/uwsgi/SCGI/gRPC 服务器接收到的响应进行转换, 当响应中已经带有“Content-Type”响应头字段中的字符集时。如果启用转换, 则使用接收到的响应中指定的字符集作为源字符集。

### **i** 备注

如果响应是在子请求中接收到的, 则无论 `override_charset` 指令设置如何, 总是会执行从响应字符集到主请求字符集的转换。

## source\_charset

语法	<code>source_charset encoding;</code>
默认值	—
上下文	http, server, location, if in location

定义响应的源字符集。如果该字符集与 `charset` 指令中指定的字符集不同, 则会进行转换。

## DAV

该模块旨在通过 WebDAV 协议实现文件管理自动化。模块处理 HTTP 和 WebDAV 方法: PUT、DELETE、MKCOL、COPY 和 MOVE。

当从源代码构建时, 默认不包含该模块; 需要通过 `--with-http_dav_module` 构建选项来启用。

在来自我们的仓库的包和镜像中, 该模块包含在构建中。

### **ii** 重要

需要其他 WebDAV 方法才能操作的 WebDAV 客户端将无法与此模块一起使用。

## 配置示例

```
location / {
    root                /data/www;

    client_body_temp_path /data/client_temp;

    dav_methods PUT DELETE MKCOL COPY MOVE;

    create_full_put_path on;
    dav_access          group:rw all:r;

    limit_except GET {
        allow 192.168.1.0/32;
        deny  all;
    }
}
```

## 指令

### create\_full\_put\_path

语法	create_full_put_path on   off;
默认值	create_full_put_path off;
上下文	http, server, location

WebDAV 规范仅允许在已存在的目录中创建文件。此指令允许创建所有需要的中间目录。

### dav\_access

语法	dav_access users:permissions ...;
默认值	dav_access user:rw;
上下文	http, server, location

为新创建的文件和目录设置访问权限, 例如:

```
dav_access user:rw group:rw all:r;
```

如果指定了任一组或所有访问权限, 则可以省略用户权限:

```
dav_access group:rw all:r;
```

## dav\_methods

语法	<code>dav_methods off   method ...;</code>
默认值	<code>dav_methods off;</code>
上下文	http, server, location

允许指定的 HTTP 和 WebDAV 方法。参数 `off` 拒绝所有此模块处理的方法。支持以下方法: PUT、DELETE、MKCOL、COPY 和 MOVE。

使用 PUT 方法上传的文件首先被写入临时文件, 然后文件被重命名。从版本 0.8.9 开始, 临时文件和持久存储可以放在不同的文件系统上。但请注意, 在这种情况下, 文件是在两个文件系统之间复制, 而不是便宜的重命名操作。因此建议在任何给定的 *location* 中, 保存的文件和由 `client_body_temp_path` 指令设置的临时文件目录放在同一个文件系统中。

使用 PUT 方法创建文件时, 可以通过在“Date”头字段中传递修改日期来指定。

## min\_delete\_depth

语法	<code>min_delete_depth number;</code>
默认值	<code>min_delete_depth 0;</code>
上下文	http, server, location

允许 DELETE 方法删除文件, 前提是请求路径中的元素数量不小于指定的数量。例如, 指令

```
min_delete_depth 4;
```

允许删除以下请求的文件

```
/users/00/00/name
/users/00/00/name/pic.jpg
/users/00/00/page.html
```

并拒绝删除

```
/users/00/00
```

## 空白 GIF

该模块发出单像素透明 GIF。

## 配置示例

```
location = /_gif {  
    empty_gif;  
}
```

## 指令

### empty\_gif

语法	empty_gif;
默认	—
上下文	location

在周围的位置中启用模块处理。

## FastCGI

该模块允许将请求传递给 FastCGI 服务器。

## 配置示例

```
location / {  
    fastcgi_pass localhost:9000;  
    fastcgi_index index.php;  
  
    fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;  
    fastcgi_param QUERY_STRING $query_string;  
    fastcgi_param REQUEST_METHOD $request_method;  
    fastcgi_param CONTENT_TYPE $content_type;  
    fastcgi_param CONTENT_LENGTH $content_length;  
}
```

## 指令

### fastcgi\_bind

语法	<code>fastcgi_bind address [transparent]   off;</code>
默认值	—
上下文	http, server, location

使与 FastCGI 服务器的出站连接从指定的本地 IP 地址（可选端口）发起。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 `fastcgi_bind` 指令的效果，这允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许出站连接从非本地 IP 地址发起，例如，从客户端的真实 IP 地址：

```
fastcgi_bind $remote_addr transparent;
```

为了使该参数生效，Angie 工作进程通常需要以超级用户 权限运行。在 Linux 上，这不是必需的：如果指定了 `transparent` 参数，工作进程将继承主进程的 `CAP_NET_RAW` 能力。

#### 重要

还应该配置内核路由表以拦截来自 FastCGI 服务器的网络流量。

### fastcgi\_buffer\_size

语法	<code>fastcgi_buffer_size size;</code>
默认值	<code>fastcgi_buffer_size 4k 8k;</code>
上下文	http, server, location

设置用于读取从 FastCGI 服务器接收的响应的第一部分的缓冲区大小。此部分通常包含一个小的响应头。默认情况下，缓冲区大小等于一个内存页。这是 4K 或 8K，具体取决于平台。它可以变得更小。

### fastcgi\_buffering

语法	<code>fastcgi_buffering on   off;</code>
默认值	<code>fastcgi_buffering on;</code>
上下文	http, server, location

启用或禁用从 FastCGI 服务器的响应的缓冲。

off	Angie 尽可能快地从 FastCGI 服务器接收响应, 并将其保存到由 <code>fastcgi_buffer_size</code> 和 <code>fastcgi_buffers</code> 指令设置的缓冲区中。如果整个响应无法容纳在内存中, 其一部分可以保存到磁盘上的临时文件中。写入临时文件由 <code>fastcgi_max_temp_file_size</code> 和 <code>fastcgi_temp_file_write_size</code> 指令控制。
on	响应在接收到时立即同步传递给客户端。Angie 不会尝试从 FastCGI 服务器读取整个响应。Angie 一次可以从服务器接收的数据的最大大小由 <code>fastcgi_buffer_size</code> 指令设置。

可以通过在“X-Accel-Buffering”响应头字段中传递“yes”或“no”来启用或禁用缓冲。此功能可以使用 `fastcgi_ignore_headers` 指令禁用。

### fastcgi\_buffers

语法	<code>fastcgi_buffers number size;</code>
默认值	<code>fastcgi_buffers 8 4k 8k;</code>
上下文	http, server, location

设置用于读取从 FastCGI 服务器接收的响应的缓冲区的数量和大小, 对于单个连接。

默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 具体取决于平台。

### fastcgi\_busy\_buffers\_size

语法	<code>fastcgi_busy_buffers_size size;</code>
默认值	<code>fastcgi_busy_buffers_size 8k   16k;</code>
上下文	http, server, location

当启用了从 FastCGI 服务器的响应的 `buffering` 时, 限制可以繁忙地向客户端发送响应的缓冲区总大小, 而响应尚未完全读取。同时, 其余的缓冲区可以用于读取响应, 并在需要时将响应的一部分缓冲到临时文件中。默认情况下, 大小受 `fastcgi_buffer_size` 和 `fastcgi_buffers` 指令设置的两个缓冲区的大小限制。

### fastcgi\_cache

语法	<code>fastcgi_cache zone   off;</code>
默认值	<code>fastcgi_cache off;</code>
上下文	http, server, location

定义用于缓存的共享内存区域。同一区域可以在多个地方使用。参数值可以包含变量。off 参数禁用从先前配置级别继承的缓存。

### fastcgi\_cache\_background\_update

语法	<code>fastcgi_cache_background_update on   off;</code>
默认值	<code>fastcgi_cache_background_update off;</code>
上下文	http, server, location

允许启动后台子请求以更新过期的缓存项, 同时将陈旧的缓存响应返回给客户端。请注意, 有必要允许在更新时使用陈旧的缓存响应。

### fastcgi\_cache\_bypass

语法	<code>fastcgi_cache_bypass <i>string</i> ...;</code>
默认值	—
上下文	http, server, location

定义在什么条件下响应将不从缓存中获取。如果字符串参数的至少一个值不为空且不等于”0”, 则响应将不从缓存中获取:

```
fastcgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;  
fastcgi_cache_bypass $http_pragma $http_authorization;
```

可以与 `fastcgi_no_cache` 指令一起使用。

### fastcgi\_cache\_key

语法	<code>fastcgi_cache_key <i>string</i>;</code>
默认值	—
上下文	http, server, location

定义缓存的键, 例如:

```
fastcgi_cache_key localhost:9000$request_uri;
```

### fastcgi\_cache\_lock

语法	<code>fastcgi_cache_lock on   off;</code>
默认值	<code>fastcgi_cache_lock off;</code>
上下文	http, server, location

启用时, 只允许一次请求通过向 FastCGI 服务器传递请求来填充根据 `fastcgi_cache_key` 指令标识的新缓存元素。其他相同缓存元素的请求将等待响应出现在缓存中或该元素的缓存锁释放, 最长时间由 `fastcgi_cache_lock_timeout` 指令设置。

### fastcgi\_cache\_lock\_age

语法	<code>fastcgi_cache_lock_age time;</code>
默认值	<code>fastcgi_cache_lock_age 5s;</code>
上下文	http, server, location

如果发送到 FastCGI 服务器以填充新缓存条目的最后一个请求在指定时间内未完成, 则可以向 FastCGI 服务器发送另一个请求。

### fastcgi\_cache\_lock\_timeout

语法	<code>fastcgi_cache_lock_timeout time;</code>
默认值	<code>fastcgi_cache_lock_timeout 5s;</code>
上下文	http, server, location

为 `fastcgi_cache_lock` 设置超时时间。当时间到期时, 请求将被传递到 FastCGI 服务器, 但是, 响应将不会被缓存。

### fastcgi\_cache\_max\_range\_offset

语法	<code>fastcgi_cache_max_range_offset number;</code>
默认值	—
上下文	http, server, location

为字节范围请求设置偏移量 (以字节为单位)。如果范围超出偏移量, 范围请求将被传递到 FastCGI 服务器, 响应将不会被缓存。



### fastcgi\_cache\_methods

语法	<code>fastcgi_cache_methods GET   HEAD   POST ...;</code>
默认值	<code>fastcgi_cache_methods GET HEAD;</code>
上下文	http, server, location

如果客户端请求方法在此指令中列出, 则响应将被缓存。”GET” 和”HEAD” 方法总是被添加到列表中, 虽然建议显式指定它们。另请参阅`fastcgi_no_cache` 指令。

### fastcgi\_cache\_min\_uses

语法	<code>fastcgi_cache_min_uses number;</code>
默认值	<code>fastcgi_cache_min_uses 1;</code>
上下文	http, server, location

设置请求次数, 在此之后响应将被缓存。

### fastcgi\_cache\_path

语法	<code>fastcgi_cache_path path [levels=levels] [use_temp_path=on   off] keys_zone=name:size [inactive=time] [max_size=size] [min_free=size] [manager_files=number] [manager_sleep=time] [manager_threshold=time] [loader_files=number] [loader_sleep=time] [loader_threshold=time];</code>
默认	—
上下文	http, server, location

设置缓存的路径和其他参数。缓存数据存储在文件中。缓存中的键和文件名均是通过代理的 URL 应用 MD5 函数得到的结果。

`levels` 参数定义了缓存的层次结构: 从 1 到 3, 每个级别接受值 1 或 2。例如, 在以下配置中

```
fastcgi_cache_path /data/angie/cache levels=1:2 keys_zone=one:10m;
```

缓存中文件名将如下所示:

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存的响应首先被写入临时文件, 然后将其重命名。临时文件和缓存可以放在不同的文件系统上。但是, 要注意在这种情况下, 文件是在两个文件系统之间复制, 而不是进行廉价的重命名操作。因此, 建议在任何给定位置, 缓存和临时文件目录都放在同一个文件系统上。

临时文件的目录是根据 `use_temp_path` 参数设置的。

on	如果省略此参数或将其设置为“on”，则会使用由 <code>fastcgi_temp_path</code> 指令为给定 <code>location</code> 设置的目录。
off	临时文件将直接放入缓存目录中。

此外，所有活动键和数据信息均存储在共享内存区域中，该区域的名称和大小由 `keys_zone` 参数配置。一个兆字节的区域可以存储约 8 千个键。

在 `inactive` 参数指定的时间内未访问的缓存数据将被移除，无论其新鲜度如何。

默认情况下，`inactive` 设置为 10 分钟。

一个特殊的 **缓存管理器** 进程监控缓存的最大大小和文件系统上的最小可用空间。当大小超过限制或空间不足时，它会移除最近最少使用的数据。数据是通过迭代方式移除的。

<code>max_size</code>	最大缓存大小
<code>min_free</code>	文件系统中缓存的最小可用空间
<code>manager_files</code>	限制一次迭代中要删除的项目数量 默认情况下，100。
<code>manager_threshold</code>	限制一次迭代的持续时间 默认情况下，200 毫秒
<code>manager_sleep</code>	配置交互之间的暂停时间 默认情况下，50 毫秒

Angie 启动一分钟后，特殊的 **缓存加载器** 进程被激活。它将文件系统上存储的先前缓存数据的信息加载到缓存区中。加载也是通过迭代方式进行的。

<code>loader_files</code>	限制一次迭代中要加载的项目数量 默认情况下，100
<code>loader_threshold</code>	限制一次迭代的持续时间 默认情况下，200 毫秒
<code>loader_sleep</code>	配置交互之间的暂停时间 默认情况下，50 毫秒

### fastcgi\_cache\_revalidate

语法	<code>fastcgi_cache_revalidate on   off;</code>
默认值	<code>fastcgi_cache_revalidate off;</code>
上下文	<code>http, server, location</code>

启用使用”If-Modified-Since” 和”If-None-Match” 头字段的条件请求来重新验证过期的缓存项。

### fastcgi\_cache\_use\_stale

语法	<code>fastcgi_cache_use_stale error   timeout   invalid_header   updating   http_500   http_503   http_403   http_429   off ...;</code>
默认值	<code>fastcgi_cache_use_stale off;</code>
上下文	<code>http, server, location</code>

确定在与 FastCGI 服务器通信时出现错误的情况下, 何时可以使用陈旧的缓存响应。此指令的参数与 `fastcgi_next_upstream` 指令的参数匹配。

<code>error</code>	允许在无法选择 FastCGI 服务器来处理请求时使用陈旧的缓存响应。
<code>updating</code>	额外参数, 允许在缓存数据正在更新时使用陈旧的缓存响应。这可以在更新缓存数据时最小化对 FastCGI 服务器的访问次数。

使用陈旧的缓存响应也可以直接在响应头中启用, 指定响应变陈旧后的秒数。

- `stale-while-revalidate` 扩展了“Cache-Control”头字段, 允许在响应正在更新时使用陈旧的缓存响应。
- `stale-if-error` 扩展了“Cache-Control”头字段, 允许在出现错误时使用陈旧的缓存响应。

#### 备注

此优先级低于使用指令参数。

为了在填充新缓存元素时最小化对 FastCGI 服务器的访问次数, 可以使用 `fastcgi_cache_lock` 指令。

### fastcgi\_cache\_valid

语法	<code>fastcgi_cache_valid [code ...] time;</code>
默认	—
上下文	<code>http, server, location</code>

设置不同响应代码的缓存时间。例如, 以下指令

```
fastcgi_cache_valid 200 302 10m;  
fastcgi_cache_valid 404 1m;
```

为响应代码 200 和 302 设置 10 分钟的缓存时间, 为响应代码 404 设置 1 分钟的缓存时间。

如果仅指定缓存时间

```
fastcgi_cache_valid 5m;
```

则仅缓存 200、301 和 302 的响应。

此外, 可以指定 `any` 参数以缓存任何响应:

```
fastcgi_cache_valid 200 302 10m;  
fastcgi_cache_valid 301      1h;  
fastcgi_cache_valid any      1m;
```

### **i** 备注

缓存参数也可以直接在响应头中设置。其优先级高于使用指令设置的缓存时间。

- "X-Accel-Expires" 头字段设置响应的缓存时间 (以秒为单位)。零值禁用响应的缓存。如果值以 @ 前缀开头, 则设置从纪元时间开始的绝对时间 (以秒为单位), 直到响应可能被缓存。
- 如果头部不包括 "X-Accel-Expires" 字段, 则缓存参数可以在 "Expires" 或 "Cache-Control" 头字段中设置。
- 如果头部包括 "Set-Cookie" 字段, 则此类响应将不被缓存。
- 如果头部包括特殊值 "\*" 的 "Vary" 字段, 则此类响应将不被缓存。如果头部包括 "Vary" 字段的其他值, 则此类响应将根据相应的请求头字段进行缓存。

可以使用 `fastcgi_ignore_headers` 指令禁用对一个或多个这些响应头字段的处理。

## fastcgi\_catch\_stderr

语法	<code>fastcgi_catch_stderr string;</code>
默认	—
上下文	http, server, location

设置在从 FastCGI 服务器接收到的响应的错误流中搜索的字符串。如果找到该字符串, 则认为 FastCGI 服务器返回了无效响应。这允许在 Angie 中处理应用程序错误, 例如:

```
location /php/ {  
    fastcgi_pass backend:9000;  
    ...  
    fastcgi_catch_stderr "PHP Fatal error";  
    fastcgi_next_upstream error timeout invalid_header;  
}
```

### fastcgi\_connect\_timeout

语法	<code>fastcgi_connect_timeout time;</code>
默认	<code>fastcgi_connect_timeout 60s;</code>
上下文	http, server, location

定义与 FastCGI 服务器建立连接的超时时间。需要注意的是, 此超时时间通常不能超过 75 秒。

### fastcgi\_connection\_drop

语法	<code>fastcgi_connection_drop time   on   off;</code>
默认	<code>fastcgi_connection_drop off;</code>
上下文	http, server, location

启用在代理服务器从组中移除或被 *reresolve* 进程或 API 命令 DELETE 标记为永久不可用后终止与其的所有连接。

当为客户端或代理服务器处理下一次读写事件时, 连接将终止。

设置 *time* 启用连接终止 超时; 设置 *on* 时, 连接会立即断开。

### fastcgi\_force\_ranges

语法	<code>fastcgi_force_ranges on   off;</code>
默认	<code>fastcgi_force_ranges off;</code>
上下文	http, server, location

无论这些响应中的“Accept-Ranges”字段如何, 均启用来自 FastCGI 服务器的缓存和非缓存响应的字节范围支持。

### fastcgi\_hide\_header

语法	<code>fastcgi_hide_header field;</code>
默认	—
上下文	http, server, location

默认情况下, Angie 不会将 FastCGI 服务器响应中的“Status”和“X-Accel-...”头字段传递给客户端。*fastcgi\_hide\_header* 指令设置不传递的其他字段。如果需要允许字段传递, 可以使用 *fastcgi\_pass\_header* 指令。

### fastcgi\_ignore\_client\_abort

语法	<code>fastcgi_ignore_client_abort on   off;</code>
默认值	<code>fastcgi_ignore_client_abort off;</code>
上下文	http, server, location

决定当客户端在未等待响应时关闭连接时, 是否应关闭与 FastCGI 服务器的连接。

### fastcgi\_ignore\_headers

语法	<code>fastcgi_ignore_headers field;</code>
默认值	—
上下文	http, server, location

禁用来自 FastCGI 服务器的某些响应头字段的处理。以下字段可以被忽略: “X-Accel-Redirect”、“X-Accel-Expires”、“X-Accel-Limit-Rate”、“X-Accel-Buffering”、“X-Accel-Charset”、“Expires”、“Cache-Control”、“Set-Cookie” 和 “Vary”。

如果未禁用, 这些头字段的处理将产生以下效果:

- “X-Accel-Expires”、“Expires”、“Cache-Control”、“Set-Cookie” 和 “Vary” 设置响应缓存的参数;
- “X-Accel-Redirect” 执行到指定 URI 的内部重定向;
- “X-Accel-Limit-Rate” 设置向客户端传输响应的速率限制;
- “X-Accel-Buffering” 启用或禁用响应的缓冲;
- “X-Accel-Charset” 设置响应的所需:ref:‘ < charset> ‘。

### fastcgi\_index

语法	<code>fastcgi_index name;</code>
默认值	—
上下文	http, server, location

设置将在 URI 以斜杠结尾时附加在 `$fastcgi_script_name` 变量值之后的文件名。例如, 使用以下设置

```
fastcgi_index index.php;  
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```

对于 `/page.php` 请求, `SCRIPT_FILENAME` 参数将等于 `/home/www/scripts/php/page.php`,

对于 `/` 请求, 它将等于 `/home/www/scripts/php/index.php`。

### fastcgi\_intercept\_errors

语法	<code>fastcgi_intercept_errors on   off;</code>
默认值	<code>fastcgi_intercept_errors off;</code>
上下文	http, server, location

决定是否应该将代码大于或等于 300 的 FastCGI 服务器响应传递给客户端, 或者拦截并重定向到 Angie 以使用 `error_page` 指令进行处理。

### fastcgi\_keep\_conn

语法	<code>fastcgi_keep_conn on   off;</code>
默认值	<code>fastcgi_keep_conn off;</code>
上下文	http, server, location

默认情况下, FastCGI 服务器会在发送响应后立即关闭连接。然而, 当该指令设置为 `on` 时, Angie 将指示 FastCGI 服务器保持连接打开。这对于使 FastCGI 服务器上的保持活动连接功能正常工作是必要的。

### fastcgi\_limit\_rate

语法	<code>fastcgi_limit_rate rate;</code>
默认值	<code>fastcgi_limit_rate 0;</code>
上下文	http, server, location

限制从 FastCGI 服务器读取响应的速度。 `rate` 以每秒字节数指定, 并可以包含变量。

0	禁用速率限制
---	--------

#### 备注

限制是针对每个请求设置的, 因此如果 Angie 同时打开两个到 FastCGI 服务器的连接, 整体速率将是指定限制的两倍。此限制仅在启用 FastCGI 服务器响应的缓冲时生效。

### fastcgi\_max\_temp\_file\_size

语法	<code>fastcgi_max_temp_file_size size;</code>
默认值	<code>fastcgi_max_temp_file_size 1024m;</code>
上下文	http, server, location

当启用了来自 FastCGI 服务器的响应的缓冲, 并且整个响应不适合由 `fastcgi_buffer_size` 和 `fastcgi_buffers` 指令设置的缓冲区时, 可以将部分响应保存到临时文件。此指令设置临时文件的最大大小。一次写入临时文件的数据大小由 `fastcgi_temp_file_write_size` 指令设置。

0	禁用对临时文件的响应缓冲
---	--------------

#### 备注

此限制不适用于将被缓存 或存储在磁盘 上的响应。

### fastcgi\_next\_upstream

语法	<code>fastcgi_next_upstream error   timeout   invalid_header   http_500   http_503   http_403   http_404   http_429   non_idempotent   off ...;</code>
默认值	<code>fastcgi_next_upstream error timeout;</code>
上下文	http, server, location

指定在何种情况下请求应传递到下一个服务器:

<code>error</code>	在与服务器建立连接、传递请求或读取响应头时发生错误;
<code>timeout</code>	在与服务器建立连接、传递请求或读取响应头时发生超时;
<code>invalid_header</code>	服务器返回空的或无效的响应;
<code>http_500</code>	服务器返回代码为 500 的响应;
<code>http_503</code>	服务器返回代码为 503 的响应;
<code>http_403</code>	服务器返回代码为 403 的响应;
<code>http_404</code>	服务器返回代码为 404 的响应;
<code>http_429</code>	服务器返回代码为 429 的响应;
<code>non_idempotent</code>	通常, 具有 非幂等 方法 (POST, LOCK, PATCH) 的请求在请求已发送到上游服务器时不会传递到下一个服务器; 启用此选项明确允许重试此类请求;
<code>off</code>	禁止将请求传递到下一个服务器。



**i 备注**

应注意, 将请求传递到下一个服务器仅在尚未向客户端发送任何内容时才可能。也就是说, 如果在响应传输的中间发生错误或超时, 修复这是不可能的。

该指令还定义了与服务器通信时被认为是不成功尝试 的情况。

error	总是被认为是不成功的尝试, 即使它们未在指令中指定
timeout	
invalid_header	
http_500	仅在指令中指定时才被认为是不成功的尝试
http_503	
http_429	
http_403	从不被认为是不成功的尝试
http_404	

将请求传递到下一个服务器可以通过[尝试次数](#) 和[时间](#) 进行限制。

**fastcgi\_next\_upstream\_timeout**

语法	<code>fastcgi_next_upstream_timeout time;</code>
默认值	<code>fastcgi_next_upstream_timeout 0;</code>
上下文	http, server, location

限制请求可以传递到下一个服务器 的时间。

0	关闭此限制
---	-------

**fastcgi\_next\_upstream\_tries**

语法	<code>fastcgi_next_upstream_tries number;</code>
默认值	<code>fastcgi_next_upstream_tries 0;</code>
上下文	http, server, location

限制将请求传递到下一个服务器 的可能尝试次数。

0	关闭此限制
---	-------

## fastcgi\_no\_cache

语法	<code>fastcgi_no_cache string ...;</code>
默认值	—
上下文	http, server, location

定义在何种条件下响应将不会被保存到缓存中。如果字符串参数的至少一个值不为空且不等于“0”，则响应不会被保存：

```
fastcgi_no_cache $cookie_nocache $arg_nocache$arg_comment;  
fastcgi_no_cache $http_pragma $http_authorization;
```

可以与 `fastcgi_cache_bypass` 指令一起使用。

## fastcgi\_param

语法	<code>fastcgi_param parameter value [if_not_empty];</code>
默认值	—
上下文	http, server, location

设置应传递给 FastCGI 服务器的参数。值可以包含文本、变量及其组合。只有在当前级别没有定义 `fastcgi_param` 指令时，这些指令才会从上一级配置继承。

以下示例显示了 PHP 的最低要求设置：

```
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;  
fastcgi_param QUERY_STRING $query_string;
```

SCRIPT\_FILENAME 参数用于 PHP 确定脚本名称，而 QUERY\_STRING 参数用于传递请求参数。

对于处理 POST 请求的脚本，还需要以下三个参数：

```
fastcgi_param REQUEST_METHOD $request_method;  
fastcgi_param CONTENT_TYPE $content_type;  
fastcgi_param CONTENT_LENGTH $content_length;
```

如果 PHP 是使用 `--enable-force-cgi-redirect` 配置参数构建的，还应传递 REDIRECT\_STATUS 参数，值为“200”：

```
fastcgi_param REDIRECT_STATUS 200;
```

如果指令指定为 `if_not_empty`，那么这样的参数将仅在其值不为空时传递给服务器：

```
fastcgi_param HTTPS          $https if_not_empty;
```

### fastcgi\_pass

语法	<code>fastcgi_pass address;</code>
默认值	—
上下文	location, if in location

设置 FastCGI 服务器的地址。地址可以指定为域名或 IP 地址, 以及端口:

```
fastcgi_pass localhost:9000;
```

或者作为 UNIX 域套接字路径:

```
fastcgi_pass unix:/tmp/fastcgi.socket;
```

如果域名解析为多个地址, 则所有地址将以轮询方式使用。此外, 地址可以指定为服务器组。如果使用了一个组, 则不能为其指定端口; 相反, 应为组中的每个服务器单独指定端口。

参数值可以包含变量。在这种情况下, 如果地址指定为域名, 则在描述的服务器组中查找该名称, 如果未找到, 则使用解析器确定。

### fastcgi\_pass\_header

语法	<code>fastcgi_pass_header field;</code>
默认值	—
上下文	http, server, location

允许从 FastCGI 服务器向客户端传递 *otherwise disabled* 头字段。

### fastcgi\_pass\_request\_body

语法	<code>fastcgi_pass_request_body on   off;</code>
默认值	—
上下文	http, server, location

指示是否将原始请求体传递给 FastCGI 服务器。另请参见 `fastcgi_pass_request_headers` 指令。

### fastcgi\_pass\_request\_headers

语法	<code>fastcgi_pass_request_headers on   off;</code>
默认值	—
上下文	http, server, location

指示是否将原始请求的头字段传递给 FastCGI 服务器。另请参见 `fastcgi_pass_request_body` 指令。

### fastcgi\_read\_timeout

语法	<code>fastcgi_read_timeout time;</code>
默认值	<code>fastcgi_read_timeout 60s;</code>
上下文	http, server, location

定义从 FastCGI 服务器读取响应的超时时间。超时仅在两次连续读取操作之间设置，而不是用于整个响应的传输。如果 FastCGI 服务器在此时间内未传输任何内容，则连接将关闭。

### fastcgi\_request\_buffering

语法	<code>fastcgi_request_buffering on   off;</code>
默认值	<code>fastcgi_request_buffering on;</code>
上下文	http, server, location

启用或禁用客户端请求体的缓冲。

on	在将请求发送到 FastCGI 服务器之前，整个请求体将从客户端读取。
off	请求体在接收到时立即发送到 FastCGI 服务器。在这种情况下，如果请求已经开始发送请求体，则无法将请求传递给下一个服务器。

### fastcgi\_send\_lowat

语法	<code>fastcgi_send_lowat size;</code>
默认值	<code>fastcgi_send_lowat 0;</code>
上下文	http, server, location

如果将指令设置为非零值，Nginx 将尝试通过使用 `kqueue` 方法的 `NOTE_LOWAT` 标志或 `SO_SNDLOWAT` 套接字选项，以指定的大小，最小化对 FastCGI 服务器的传出连接上的发送操

作次数。

#### **i** 备注

在 Linux、Solaris 和 Windows 上忽略此指令。

### fastcgi\_send\_timeout

语法	<code>fastcgi_send_timeout time;</code>
默认值	<code>fastcgi_send_timeout 60s;</code>
上下文	http, server, location

设置将请求传输到 FastCGI 服务器的超时时间。超时仅在两次连续写操作之间设置，而不是用于整个请求的传输。如果 FastCGI 服务器在此时间内未接收到任何内容，则连接将关闭。

### fastcgi\_socket\_keepalive

语法	<code>fastcgi_socket_keepalive on   off;</code>
默认值	<code>fastcgi_socket_keepalive off;</code>
上下文	http, server, location

配置与 FastCGI 服务器的传出连接的“TCP 保持活动”行为。

""	默认情况下，操作系统的设置对套接字生效。
on	为套接字启用 <code>SO_KEEPALIVE</code> 套接字选项。

### fastcgi\_split\_path\_info

语法	<code>fastcgi_split_path_info regex;</code>
默认值	—
上下文	location

定义一个正则表达式，用于捕获 `$fastcgi_path_info` 变量的值。正则表达式应有两个捕获：第一个成为 `$fastcgi_script_name` 变量的值，第二个成为 `$fastcgi_path_info` 变量的值。例如，使用以下设置

```
location ~ ^(\.\.php)(.*)$ {
    fastcgi_split_path_info    ^(\.\.php)(.*)$;
```

```
fastcgi_param SCRIPT_FILENAME /path/to/php$fastcgi_script_name;  
fastcgi_param PATH_INFO      $fastcgi_path_info;
```

对于 `/show.php/article/0001` 请求, `SCRIPT_FILENAME` 参数将等于 `/path/to/php/show.php`,  
`PATH_INFO` 参数将等于 `/article/0001`。

### fastcgi\_store

语法	<code>fastcgi_store on   off   <i>string</i>;</code>
默认值	<code>fastcgi_store off;</code>
上下文	<code>http, server, location</code>

启用将文件保存到磁盘。

<code>on</code>	使用与指令 <code>alias</code> 或 <code>root</code> 对应的路径保存文件。
<code>off</code>	禁用文件保存

此外, 文件名可以使用带有变量的字符串显式设置:

```
fastcgi_store /data/www$original_uri;
```

文件的修改时间根据接收到的“Last-Modified”响应头字段设置。响应首先写入临时文件, 然后文件被重命名。临时文件和持久存储可以放在不同的文件系统上。然而, 请注意, 在这种情况下, 文件是在两个文件系统之间复制的, 而不是廉价的重命名操作。因此, 建议为任何给定的 `location`, 保存的文件和由 `fastcgi_temp_path` 指令设置的保存临时文件的目录都放在同一文件系统上。

此指令可用于创建静态不可变文件的本地副本, 例如:

```
location /images/ {  
    root                /data/www;  
    error_page          404 = /fetch$uri;  
}  
  
location /fetch/ {  
    internal;  
  
    fastcgi_pass        backend:9000;  
    ...  
  
    fastcgi_store       on;  
    fastcgi_store_access user:rw group:rw all:r;  
    fastcgi_temp_path   /data/temp;
```

```
alias /data/www/;
}
```

### fastcgi\_store\_access

语法	<code>fastcgi_store_access users:permissions ...;</code>
默认值	<code>fastcgi_store_access user:rw;</code>
上下文	http, server, location

设置新创建文件和目录的访问权限, 例如:

```
fastcgi_store_access user:rw group:rw all:r;
```

如果指定了任何组或所有访问权限, 则用户权限可以省略:

```
fastcgi_store_access group:rw all:r;
```

### fastcgi\_temp\_file\_write\_size

语法	<code>fastcgi_temp_file_write_size size;</code>
默认值	<code>fastcgi_temp_file_write_size 8k 16k;</code>
上下文	http, server, location

限制在启用从 FastCGI 服务器缓冲响应到临时文件时, 一次写入临时文件的数据大小。默认情况下, 大小由 `fastcgi_buffer_size` 和 `fastcgi_buffers` 指令设置的两个缓冲区限制。临时文件的最大大小由 `fastcgi_max_temp_file_size` 指令设置。

### fastcgi\_temp\_path

语法	<code>fastcgi_temp_path path [level1 [level2 [level3]]];</code>
默认值	<code>fastcgi_temp_path fastcgi_temp;</code> (路径取决于 <code>--http-fastcgi-temp-path</code> 构建选项)
上下文	http, server, location

定义存储从 FastCGI 服务器接收到的数据的临时文件的目录。可以在指定目录下使用最多三层子目录层次结构。例如, 在以下配置中

```
fastcgi_temp_path /spool/angie/fastcgi_temp 1 2;
```

临时文件可能如下所示:

```
/spool/angie/fastcgi_temp/7/45/00000123457
```

另请参见 `fastcgi_cache_path` 指令的 `use_temp_path` 参数。

### 传递给 FastCGI 服务器的参数

HTTP 请求头字段作为参数传递给 FastCGI 服务器。在作为 FastCGI 服务器运行的应用程序和脚本中, 这些参数通常作为环境变量提供。例如, "User-Agent" 请求头字段作为 `HTTP_USER_AGENT` 参数传递。除了 HTTP 请求头字段, 还可以使用 `fastcgi_param` 指令传递任意参数。

### 内置变量

`http_fastcgi` 模块支持内置变量, 可以使用 `fastcgi_param` 指令来设置参数:

```
$fastcgi_script_name
```

请求 URI 或者, 如果 URI 以斜杠结尾, 则为附加了由 `fastcgi_index` 指令配置的索引文件名的请求 URI。此变量可以用于设置 PHP 中确定脚本名称的 `SCRIPT_FILENAME` 和 `PATH_TRANSLATED` 参数。例如, 对于 `/info/` 请求, 使用以下指令

```
fastcgi_index index.php;  
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```

`SCRIPT_FILENAME` 参数将等于 `/home/www/scripts/php/info/index.php`。

使用 `fastcgi_split_path_info` 指令时, `$fastcgi_script_name` 变量等于指令设置的第一个捕获组的值。

```
$fastcgi_path_info
```

由 `fastcgi_split_path_info` 指令设置的第二个捕获组的值。此变量可用于设置 `PATH_INFO` 参数。

### FLV

该模块为 Flash 视频 (FLV) 文件提供伪流式传输的服务器端支持。

它专门处理请求 URI 的查询字符串中的 `start` 参数, 通过从请求的字节偏移量开始发送文件内容, 并附加 FLV 头。

当从源代码构建时, 该模块默认未构建; 需要使用 `--with-http_flv_module` 构建选项来启用。

在来自我们的仓库的软件包和镜像中, 该模块已包含在构建中。



## 配置示例

```
location ~ /\.flv$ {  
    flv;  
}
```

## 指令

### flv

语法	flv;
默认值	—
上下文	location

在周围的 location 中开启模块处理。

## Geo

该模块根据客户端 IP 地址创建具有不同值的变量。

## 配置示例

```
geo $geo {  
    default          0;  
  
    127.0.0.1        2;  
    192.168.1.0/24  1;  
    10.1.0.0/16     1;  
  
    ::1              2;  
    2001:0db8::/32  1;  
}
```

## 指令

### geo

Syntax	geo [ <i>\$address</i> ] <i>\$variable</i> { ... }
Default	—
Context	http

描述了指定变量的值如何依赖于客户端 IP 地址。默认情况下, 地址从 `$remote_addr` 变量中获取, 也可以从其他变量中获取, 例如:

```
geo $arg_remote_addr $geo {  
    ...;  
}
```

### **i** 备注

由于变量仅在使用时才会被计算, 即使声明了大量的 `geo` 变量, 也不会对请求处理造成额外开销。

如果变量的值不是有效的 IP 地址, 则会使用 "255.255.255.255" 地址。

地址可以用 CIDR 表示法指定为前缀 (包括单个地址) 或作为范围。

还支持以下特殊参数:

<code>delete</code>	删除指定的网络
<code>default</code>	如果客户端地址不匹配任何指定的地址, 则为变量设置的值。当地址以 CIDR 表示法指定时, 可以使用 "0.0.0.0/0" 和 "::/0" 代替 <code>default</code> 。如果未指定 <code>default</code> , 则默认值将为空字符串。
<code>include</code>	包含一个带有地址和值的文件。可以有多个包含。
<code>proxy</code>	定义受信任的地址。当请求来自受信任的地址时, 将使用 "X-Forwarded-For" 请求头字段中的地址。与常规地址不同, 受信任的地址是按顺序检查的。
<code>proxy_recursive</code>	启用递归地址搜索。如果禁用递归搜索, 则将使用 "X-Forwarded-For" 中的最后一个地址, 而不是与受信任地址匹配的原始客户端地址。如果启用递归搜索, 则将使用 "X-Forwarded-For" 中的最后一个非受信任地址, 而不是与受信任地址匹配的原始客户端地址。
<code>ranges</code>	表示地址是作为范围指定的。此参数应为第一个。为加快地理数据库的加载, 地址应按升序排列。

示例:

```
geo $country {  
    default      ZZ;  
    include      conf/geo.conf;  
    delete      127.0.0.0/16;  
    proxy        192.168.100.0/24;  
    proxy        2001:0db8::/32;  
  
    127.0.0.0/24 US;  
    127.0.0.1/32 RU;  
    10.1.0.0/16  RU;  
    192.168.1.0/24 UK;  
}
```

`conf/geo.conf` 文件可能包含以下行:

```
10.2.0.0/16    RU;  
192.168.2.0/24 RU;
```

使用最具体的匹配值。例如, 对于 `127.0.0.1` 地址, 将选择值 `RU`, 而不是 `US`。

示例范围描述:

```
geo $country {  
    ranges;  
    default                ZZ;  
    127.0.0.0-127.0.0.0    US;  
    127.0.0.1-127.0.0.1    RU;  
    127.0.0.2-127.0.0.255  US;  
    10.1.0.0-10.1.255.255  RU;  
    192.168.1.0-192.168.1.255 UK;  
}
```

## GeoIP

根据客户端 IP 地址创建变量并赋值, 使用预编译的 `MaxMind` 数据库或其对应版本。

使用支持 IPv6 的数据库时, IPv4 地址将作为映射到 IPv6 的地址进行查找。

当从源代码构建时, 默认不构建此模块; 应通过 `--with-http_geoip_module` 构建选项进行启用。

### 重要

此模块需要 `MaxMind GeoIP` 数据库或类似的 `MaxMind GeoLite2`。

## 配置示例

```
http {  
    geoip_country    GeoIP.dat;  
    geoip_city       GeoLiteCity.dat;  
    geoip_proxy      192.168.100.0/24;  
    geoip_proxy      2001:0db8::/32;  
    geoip_proxy_recursive on;  
    ...  
}
```

## 指令

### geip\_country

语法	<code>geip_country file;</code>
默认	—
上下文	http

指定用于根据客户端 IP 地址确定国家的数据库。使用此数据库时, 可用以下变量:

<code>\$geip_country_c</code>	两位的国家代码, 例如"RU", "US"。
<code>\$geip_country_c</code>	三位的国家代码, 例如"RUS", "USA"。
<code>\$geip_country_n</code>	国家名称, 例如"Russian Federation", "United States"。

### geip\_city

语法	<code>geip_city file;</code>
默认	—
上下文	http

指定用于根据客户端 IP 地址确定国家、地区和城市的数据库。使用此数据库时, 可用以下变量:

<code>\$geip_city_cont</code>	两位的大陆代码, 例如"EU", "NA"。
<code>\$geip_city_coun</code>	两位的国家代码, 例如"RU", "US"。
<code>\$geip_city_coun</code>	三位的国家代码, 例如"RUS", "USA"。
<code>\$geip_city_coun</code>	国家名称, 例如"Russian Federation", "United States"。
<code>\$geip_dma_code</code>	美国的 DMA 区域代码 (也称为"metro code"), 根据 Google AdWords API 中的 <a href="#">地理定位</a> 。
<code>\$geip_latitude</code>	纬度。
<code>\$geip_longitude</code>	经度。
<code>\$geip_region</code>	两位的国家地区代码 (地区、领地、州、省、联邦土地等), 例如"48", "DC"。
<code>\$geip_region_na</code>	国家地区名称 (地区、领地、州、省、联邦土地等), 例如"Moscow City", "District of Columbia"。
<code>\$geip_city</code>	城市名称, 例如"Moscow", "Washington"。
<code>\$geip_postal_co</code>	邮政编码。

### geoip\_org

语法	<code>geoip_org file;</code>
默认	—
上下文	http

指定用于根据客户端 IP 地址确定组织的数据库。使用此数据库时, 可用以下变量:

<code>\$geoip_org</code>	组织名称, 例如“The University of Melbourne”。
--------------------------	--

### geoip\_proxy

语法	<code>geoip_proxy file;</code>
默认	—
上下文	http

定义可信任地址。当请求来自可信任地址时, 将使用“*X-Forwarded-For*”请求头字段中的地址。

### geoip\_proxy\_recursive

语法	<code>geoip_proxy_recursive on   off;</code>
默认	<code>geoip_proxy_recursive off;</code>
上下文	http

如果禁用递归搜索, 则将使用“*X-Forwarded-For*”中发送的最后一个地址, 而不是与可信任地址匹配的原始客户端地址。如果启用递归搜索, 则将使用“*X-Forwarded-For*”中发送的最后一个非可信任地址, 而不是与可信任地址匹配的原始客户端地址。

### gRPC

允许将请求传递给 gRPC 服务器。该模块需要 *HTTP/2*。

## 配置示例

```
server {  
    listen 9000;  
  
    http2 on;  
  
    location / {  
        grpc_pass 127.0.0.1:9000;  
    }  
}
```

## 指令

### grpc\_bind

语法	<code>grpc_bind address [transparent]   off;</code>
默认值	—
上下文	http, server, location

使与 gRPC 服务器的出站连接从指定的本地 IP 地址发起, 并可选择指定端口。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 `grpc_bind` 指令的效果, 允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许与 gRPC 服务器的出站连接从非本地 IP 地址发起, 例如来自客户端的真实 IP 地址:

```
grpc_bind $remote_addr transparent;
```

为了使此参数生效, 通常需要使用超级用户 权限运行 Angie 工作进程。在 Linux 上, 如果指定了 `transparent` 参数, 则不需要, 因为工作进程从主进程继承了 `CAP_NET_RAW` 能力。

#### 重要

必须配置内核路由表以拦截来自 gRPC 服务器的网络流量。

### grpc\_buffer\_size

语法	<code>grpc_buffer_size size;</code>
默认值	<code>grpc_buffer_size 4k 8k;</code>
上下文	http, server, location

设置用于读取从 gRPC 服务器接收到的响应的第一部分的缓冲区大小。响应在接收到后立即同步传递给客户端。

### grpc\_connect\_timeout

语法	<code>grpc_connect_timeout time;</code>
默认值	<code>grpc_connect_timeout 60s;</code>
上下文	http, server, location

定义与 gRPC 服务器建立连接的超时时间。应注意, 此超时时间通常不能超过 75 秒。

### grpc\_connection\_drop

语法	<code>grpc_connection_drop time   on   off;</code>
默认值	<code>grpc_connection_drop off;</code>
上下文	http, server, location

在代理服务器从组中移除或被重新解析过程或 API 命令 DELETE 标记为永久不可用后, 启用终止与其的所有连接。

当为客户端或代理服务器处理下一个读或写事件时, 连接将终止。

设置 *time* 启用连接终止 超时; 使用 *on* 设置时, 连接将立即中断。

### grpc\_hide\_header

语法	<code>grpc_hide_header field;</code>
默认值	—
上下文	http, server, location

默认情况下, Angie 不会将 gRPC 服务器响应中的 "Date"、"Server" 和 "X-Accel-..." 头字段传递给客户端。 `grpc_hide_header` 指令设置不会传递的其他字段。如果相反, 需要允许传递字段, 可以使用 `grpc_pass_header` 指令。

### grpc\_ignore\_headers

语法	grpc_ignore_headers <i>field</i> ...;
默认值	—
上下文	http, server, location

禁用处理来自 gRPC 服务器的某些响应头字段。可以忽略以下字段: "X-Accel-Redirect" 和 "X-Accel-Charset"。

如果未禁用, 这些头字段的处理会产生以下效果:

- "X-Accel-Redirect" 执行到指定 URI 的**内部** 重定向;
- "X-Accel-Charset" 设置响应的目标字符集。

### grpc\_intercept\_errors

语法	grpc_intercept_errors on   off;
默认值	grpc_intercept_errors off;
上下文	http, server, location

确定是否应将代码大于或等于 300 的 gRPC 响应传递给客户端, 或被拦截并重定向到 `Angie` 进行 `error_page` 指令处理。

### grpc\_next\_upstream

语法	grpc_next_upstream error   timeout   invalid_header   http_500   http_502   http_503   http_504   http_403   http_404   http_429   non_idempotent   off ...;
默认值	grpc_next_upstream error timeout;
上下文	http, server, location

指定在以下情况下请求应传递到下一个**上游池** 中的服务器:



error	与服务器建立连接、向其传递请求或读取响应头时发生错误;
timeout	与服务器建立连接、向其传递请求或读取响应头时超时;
invalid_header	服务器返回空或无效响应;
http_500	服务器返回代码 500 的响应;
http_502	服务器返回代码 502 的响应;
http_503	服务器返回代码 503 的响应;
http_504	服务器返回代码 504 的响应;
http_403	服务器返回代码 403 的响应;
http_404	服务器返回代码 404 的响应;
http_429	服务器返回代码 429 的响应;
non_idempotent	通常情况下, 使用 <b>非幂等</b> 方法 (POST, LOCK, PATCH) 的请求在向上游服务器发送请求后不会传递到下一个服务器; 启用此选项显式允许重试此类请求;
off	禁止将请求传递到下一个服务器。

**i 备注**

应记住, 仅当尚未向客户端发送任何内容时, 才有可能将请求传递到下一个服务器。也就是说, 如果在传输响应的中途发生错误或超时, 无法修复此问题。

该指令还定义了与服务器通信的**不成功尝试** 的标准。

error	timeout	始终视为不成功的尝试, 即使它们未在指令中指定
invalid_header		
http_500		仅当它们在指令中指定时才被视为不成功的尝试
http_502		
http_503		
http_504		
http_429		
http_403		永远不视为不成功的尝试
http_404		

请求传递到下一个服务器的次数可以通过**尝试次数** 和**时间** 来限制。

**grpc\_next\_upstream\_timeout**

语法	grpc_next_upstream_timeout <i>time</i> ;
默认值	grpc_next_upstream_timeout 0;
上下文	http, server, location

限制在请求可以传递到下一个 服务器的时间。

0	关闭此限制
---	-------

### grpc\_next\_upstream\_tries

语法	<code>grpc_next_upstream_tries number;</code>
默认值	<code>grpc_next_upstream_tries 0;</code>
上下文	http, server, location

限制将请求传递到下一个 服务器的尝试次数。

0	关闭此限制
---	-------

### grpc\_pass

语法	<code>grpc_pass address;</code>
默认值	—
上下文	location, 如果在 location 中

设置 gRPC 服务器地址。地址可以指定为域名或 IP 地址, 以及端口:

```
grpc_pass localhost:9000;
```

或作为 UNIX 域套接字路径:

```
grpc_pass unix:/tmp/grpc.socket;
```

或者, 可以使用”`grpc://`”方案:

```
grpc_pass grpc://127.0.0.1:9000;
```

要在 SSL 上使用 gRPC, 应使用”`grpcs://`”方案:

```
grpc_pass grpcs://127.0.0.1:443;
```

如果一个域名解析为多个地址, 则所有地址将以轮询方式使用。此外, 地址还可以指定为服务器组。如果使用组, 则不能与其一起指定端口; 相反, 应该为组内的每个服务器单独指定端口。

参数值可以包含变量。在这种情况下, 如果地址指定为域名, 则在描述的服务器组中搜索该名称, 如果未找到, 则使用解析器 确定。

### grpc\_pass\_header

语法	<code>grpc_pass_header field ...;</code>
默认值	—
上下文	http, server, location

允许传递来自 gRPC 服务器到客户端的已禁用 头字段。

### grpc\_read\_timeout

语法	<code>grpc_read_timeout 时间;</code>
默认值	<code>grpc_read_timeout 60s;</code>
上下文	http, server, location

定义从 gRPC 服务器读取响应的超时时间。此超时时间仅在两次连续读取操作之间设置，而不是针对整个响应的传输。如果 gRPC 服务器在此时间内未传输任何数据，则连接将关闭。

### grpc\_send\_timeout

语法	<code>grpc_send_timeout 时间;</code>
默认值	<code>grpc_send_timeout 60s;</code>
上下文	http, server, location

设置向 gRPC 服务器发送请求的超时时间。该超时时间仅在两次连续写操作之间设置，而不是针对整个请求的传输。如果 gRPC 服务器在此时间内未接收到任何数据，则连接将关闭。

### grpc\_set\_header

语法	<code>grpc_set_header 字段值;</code>
默认值	<code>grpc_set_header Content-Length \$content_length;</code>
上下文	http, server, location

允许重新定义或添加字段到传递给 gRPC 服务器的请求头 *passed*。值可以包含文本、变量及其组合。这些指令仅在当前级别没有定义 `grpc_set_header` 指令时，从上一级配置继承。

如果头字段的值是空字符串，则该字段将不会传递给 gRPC 服务器：

```
grpc_set_header Accept-Encoding "";
```

### grpc\_socket\_keepalive

语法	grpc_socket_keepalive on   off;
默认值	grpc_socket_keepalive off;
上下文	http, server, location

配置与 gRPC 服务器的出站连接的“TCP keepalive”行为。

""	默认情况下, 操作系统的设置对套接字生效。
on	套接字的 <i>SO_KEEPALIVE</i> 选项被打开。

### grpc\_ssl\_certificate

语法	grpc_ssl_certificate 文件;
默认值	—
上下文	http, server, location

指定用于认证 gRPC SSL 服务器的 PEM 格式的证书文件。文件名中可以使用变量。

### grpc\_ssl\_certificate\_key

语法	grpc_ssl_certificate_key 文件;
默认值	—
上下文	http, server, location

指定用于认证 gRPC SSL 服务器的 PEM 格式的私钥文件。

可以使用“engine:name:id”而不是文件, 加载来自 OpenSSL 引擎名称的指定 id 的私钥。文件名中可以使用变量。

### grpc\_ssl\_ciphers

语法	grpc_ssl_ciphers 密码套件;
默认值	grpc_ssl_ciphers DEFAULT;
上下文	http, server, location

指定对 gRPC SSL 服务器请求启用的密码套件。密码套件以 OpenSSL 库理解的格式指定。

完整列表可以通过“`openssl ciphers`”命令查看。

### `grpc_ssl_conf_command`

语法	<code>grpc_ssl_conf_command</code> 名称值;
默认值	—
上下文	http, server, location

设置建立与 gRPC SSL 服务器连接时的任意 OpenSSL 配置命令。

#### 重要

该指令在使用 OpenSSL 1.0.2 或更高版本时支持。

可以在同一级别指定多个 `grpc_ssl_conf_command` 指令。这些指令仅在当前级别没有定义 `grpc_ssl_conf_command` 指令时, 从上一级配置继承。

#### 小心

注意, 直接配置 OpenSSL 可能导致意外行为。

语法	<code>grpc_ssl_cr1</code> 文件;
默认值	—
上下文	http, server, location

指定用于验证 gRPC SSL 服务器证书的 PEM 格式的吊销证书 (CRL) 文件。

### `grpc_ssl_name`

语法	<code>grpc_ssl_name</code> 名称;
默认值	<code>grpc_ssl_name 'host' from grpc_pass;</code>
上下文	http, server, location

允许覆写用于验证 gRPC SSL 服务器证书的服务器名称, 并在与 gRPC SSL 服务器建立连接时通过 SNI 传递 *passed through SNI*。

默认情况下, 使用 `grpc_pass` URL 的主机部分。

### grpc\_ssl\_password\_file

语法	grpc_ssl_password_file 文件;
默认值	—
上下文	http, server, location

指定用于私钥 的密码短语文件, 其中每个密码短语在单独一行上指定。加载密钥时依次尝试密码短语。

### grpc\_ssl\_protocols

语法	grpc_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];
默认值	grpc_ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
上下文	http, server, location

在 1.2.0 版本发生变更: TLSv1.3 参数添加到默认设置中。

为对 gRPC HTTPS 服务器的请求启用指定的协议。

### grpc\_ssl\_server\_name

语法	grpc_ssl_server_name on   off;
默认值	grpc_ssl_server_name off;
上下文	http, server, location

启用或禁用通过 `grpc_ssl_name` 指令设定的服务器名称通过 Server Name Indication TLS 扩展 (SNI, RFC 6066) 在与 gRPC SSL 服务器建立连接时传递。

### grpc\_ssl\_session\_reuse

语法	grpc_ssl_session_reuse on   off;
默认值	grpc_ssl_session_reuse on;
上下文	http, server, location

确定是否在与 gRPC 服务器交互时可以重用 SSL 会话。如果在日志中出现 `SSL3_GET_FINISHED:digest check failed` 错误, 请尝试禁用 `session reuse`。

### grpc\_ssl\_trusted\_certificate

语法	grpc_ssl_trusted_certificate 文件;
默认值	—
上下文	http, server, location

指定用于验证 gRPC SSL 服务器证书的 PEM 格式的受信任 CA 证书文件。

### grpc\_ssl\_verify

语法	grpc_ssl_verify on   off;
默认值	grpc_ssl_verify off;
上下文	http, server, location

启用或禁用对 gRPC SSL 服务器证书的验证。

### grpc\_ssl\_verify\_depth

语法	grpc_ssl_verify_depth 数字;
默认值	grpc_ssl_verify_depth 1;
上下文	http, server, location

设置 gRPC SSL 服务器证书链的验证深度。

## GunZIP

该模块是一个过滤器, 用于为不支持“gzip”编码方法的客户端解压缩带有“Content-Encoding: gzip”的响应。当需要存储压缩数据以节省空间和减少 I/O 成本时, 该模块将非常有用。

当从源代码构建时(参见 sourcebuild), 默认情况下不会构建此模块; 应通过 `--with-http_gunzip_module` 构建选项启用。

在来自 我们的仓库的软件包和镜像中, 构建中已包含该模块。

## 配置示例

```
location /storage/ {  
    gunzip on;  
    # ...  
}
```

## 指令

### gunzip

语法	gunzip on   off;
默认值	gunzip off;
上下文	http, server, location

启用或禁用对缺乏 gzip 支持的客户端的压缩响应解压缩。如果启用, 还会考虑以下指令以确定客户端是否支持 gzip: *gzip\_http\_version*、*gzip\_proxied* 和 *gzip\_disable*。另请参阅 *gzip\_vary* 指令。

### gunzip\_buffers

语法	gunzip_buffers 数量 大小;
默认值	gunzip_buffers 32 4k   16 8k;
上下文	http, server, location

设置用于解压缩响应的缓冲区的数量和大小。默认情况下, 缓冲区大小等于一个内存页大小。这取决于平台, 可能是 4K 或 8K。

## GZip

该模块是一个使用“gzip”方法压缩响应的过滤器。这通常有助于将传输数据的大小减少一半甚至更多。

### ⚠️ 小心

使用 SSL/TLS 协议时, 压缩响应可能会受到 BREACH 攻击。



## 配置示例

```
gzip                on;  
gzip_min_length    1000;  
gzip_proxied       expired no-cache no-store private auth;  
gzip_types         text/plain application/xml;
```

可以使用 `$gzip_ratio` 变量来记录达到的压缩比率。

## 指令

### gzip

Syntax	<code>gzip on   off;</code>
默认	<code>gzip off;</code>
Context	http, server, location, if in location

启用或禁用响应的 gzip 压缩。

### gzip\_buffers

Syntax	<code>gzip_buffers number size;</code>
默认	<code>gzip_buffers 32 4k   16 8k;</code>
Context	http, server, location

设置用于压缩响应的缓冲区数量和大小。默认情况下, 缓冲区大小等于一个内存页大小, 这取决于平台, 通常是 4K 或 8K。

### gzip\_comp\_level

Syntax	<code>gzip_comp_level level;</code>
默认	<code>gzip_comp_level 1;</code>
Context	http, server, location

设置响应的 gzip 压缩级别。可接受的值范围从 1 到 9。

### gzip\_disable

Syntax	<code>gzip_disable <i>regex</i> ...;</code>
默认	—
Context	http, server, location

禁止对“User-Agent”请求头字段与指定正则表达式匹配的请求进行 gzip 压缩。

特殊的掩码 `msie6` 对应于正则表达式“`MSIE [4-6].`”，但工作速度更快。“`MSIE 6.0; ... SV1`”被排除在该掩码之外。

### gzip\_http\_version

Syntax	<code>gzip_http_version 1.0   1.1;</code>
默认	<code>gzip_http_version 1.1;</code>
Context	http, server, location

设置请求所需的最低 HTTP 版本以压缩响应。

### gzip\_min\_length

Syntax	<code>gzip_min_length <i>length</i>;</code>
默认	<code>gzip_min_length 20;</code>
Context	http, server, location

设置将被 gzip 压缩的响应的最小长度。长度仅从“Content-Length”响应头字段中确定。

### gzip\_proxied

Syntax	<code>gzip_proxied off   expired   no-cache   no-store   private   no_last_modified   no_etag   auth   any ...;</code>
默认	<code>gzip_proxied off;</code>
Context	http, server, location

根据请求和响应的情况启用或禁用对代理请求的响应进行 gzip 压缩。请求是代理请求的事实由请求头字段“Via”的存在来确定。该指令接受多个参数：

<code>off</code>	禁用对所有代理请求的压缩, 忽略其他参数;
<code>expired</code>	如果响应头包含"Expires" 字段且值禁止缓存, 则启用压缩;
<code>no-cache</code>	如果响应头包含"Cache-Control" 字段且具有"no-cache" 参数, 则启用压缩;
<code>no-store</code>	如果响应头包含"Cache-Control" 字段且具有"no-store" 参数, 则启用压缩;
<code>private</code>	如果响应头包含"Cache-Control" 字段且具有"private" 参数, 则启用压缩;
<code>no_last_modified</code>	如果响应头不包含>Last-Modified" 字段, 则启用压缩;
<code>no_etag</code>	如果响应头不包含"ETag" 字段, 则启用压缩;
<code>auth</code>	如果请求头包含"Authorization" 字段, 则启用压缩;
<code>any</code>	对所有代理请求启用压缩。

### gzip\_types

Syntax	<code>gzip_types mime-type ...;</code>
默认	<code>gzip_types text/html;</code>
Context	http, server, location

启用对指定 MIME 类型的响应进行 gzip 压缩, 除了"text/html" 之外。特殊值"\*" 匹配任何 MIME 类型。"text/html" 类型的响应始终被压缩。

### gzip\_vary

Syntax	<code>gzip_vary on   off;</code>
默认	<code>gzip_vary off;</code>
Context	http, server, location

启用或禁用在响应头中插入"Vary: Accept-Encoding" 字段, 如果指令 `gzip`、`gzip_static` 或 `gunzip` 激活。

### 内置变量

#### `$gzip_ratio`

计算出的压缩比率, 作为原始响应大小与压缩响应大小的比率。

## GZip 静态

允许发送带有“.gz”文件扩展名的预压缩文件，而不是常规文件。

当从源码构建时，该模块默认未构建；它应通过 `--with-http_gzip_static_module` 构建选项启用。

在从我们的仓库中的包和镜像中，该模块已包含在构建中。

## 配置示例

```
gzip_static on;  
gzip_proxied expired no-cache no-store private auth;
```

## 指令

### gzip\_static

语法	<code>gzip_static on   off   always;</code>
默认值	<code>gzip_static off;</code>
上下文	http, server, location

启用 (on) 或禁用 (off) 检查预压缩文件的存在。以下指令也会被考虑：`gzip_http_version`、`gzip_proxied`、`gzip_disable` 和 `gzip_vary`。

使用 `always` 时，压缩文件在所有情况下都会被使用，而无需检查客户端是否支持。这在磁盘上没有未压缩文件或者使用了 *GunZIP* 时很有用。

文件可以通过 `gzip` 命令或任何其他兼容命令进行压缩。建议原始文件和压缩文件的修改日期和时间相同。

## 头信息

允许在响应头中添加“Expires”和“Cache-Control”头字段，以及任意字段。

## 配置示例

```
expires 24h;  
expires modified +24h;  
expires @24h;  
expires 0;  
expires -1;  
expires epoch;  
expires $expires;  
add_header Cache-Control private;
```

## 指令

### add\_header

语法	<code>add_header name value [always];</code>
默认值	—
上下文	http, server, location, if in location

在响应码为 200、201、204、206、301、302、303、304、307 或 308 时, 添加指定字段到响应头。参数值可以包含变量。

可以有多个 `add_header` 指令。只有在当前级别未定义任何 `add_header` 指令时, 这些指令才会从上一级配置继承。

如果指定了 `always` 参数, 则无论响应码如何, 都会添加该头字段。

### add\_trailer

语法	<code>add_trailer name value [always];</code>
默认值	—
上下文	http, server, location, if in location

在响应码为 200、201、206、301、302、303、307 或 308 时, 添加指定字段至响应的末尾。参数值可以包含变量。

可以有多个 `add_trailer` 指令。只有在当前级别未定义任何 `add_trailer` 指令时, 这些指令才会从上一级配置继承。

如果指定了 `always` 参数, 则无论响应码如何, 都会添加该字段。

### expires

语法	<code>expires [modified] time;</code> <code>expires epoch   max   off;</code>
默认值	<code>expires off;</code>
上下文	http, server, location, if in location

启用或禁用响应码为 200、201、204、206、301、302、303、304、307 或 308 时添加或修改“Expires”和“Cache-Control”响应头字段。参数可以是正或负的时间。

“Expires”字段中的时间是当前时间与指令中指定时间的总和。如果使用了 `modified` 参数, 则时间是文件修改时间与指令中指定时间的总和。

此外, 可以使用”@”前缀指定一天中的某个时间:

```
expires @15h30m;
```

”Cache-Control”字段的内容取决于指定时间的符号:

- 时间为负—”Cache-Control: no-cache”。
- 时间为正或零—”Cache-Control: max-age=’t’”, 其中  $t$  是指令中指定的时间, 以秒为单位。

epoch	将”Expires”设置为”Thu, 01 Jan 1970 00:00:01 GMT”, ”Cache-Control”设置为”no-cache”。
max	将”Expires”设置为”Thu, 31 Dec 2037 23:55:55 GMT”, ”Cache-Control”设置为 10 年。
off	禁止添加或修改”Expires”和”Cache-Control”响应头字段。

最后的参数值可以包含变量:

```
map $sent_http_content_type $expires {  
    default          off;  
    application/pdf 42d;  
    ~image/         max;  
}  
  
expires $expires;
```

## 图像过滤器

该模块是一个过滤器, 可以转换 JPEG、GIF、PNG 和 WebP 格式的图像。

当从源代码构建时, 默认情况下不会构建此模块; 需要通过 `--with-http_image_filter_module` 构建选项启用。

在我们的代码库中, 此模块是动态构建的, 并作为一个名为 `angie-module-image-filter` 或 `angie-pro-module-image-filter` 的独立包提供。

### 重要

此模块使用 `libgd` 库。建议使用该库的最新可用版本。

要转换 WebP 格式的图像, `libgd` 库必须在编译时支持 WebP。

## 配置示例

```
location /img/ {
    proxy_pass http://backend;
    image_filter resize 150 100;
    image_filter rotate 90;
    error_page 415 = /empty;
}

location = /empty {
    empty_gif;
}
```

## 指令

### image\_filter

#### 语法

```
image_filter off;
image_filter test; image_filter size; image_filter rotate 90 | 180 | 270;
image_filter resize width height; image_filter crop width height;
```

默认值 image\_filter off;

上下文 location

设置对图像执行的转换类型:

<code>off</code>	关闭在周围位置的模块处理。
<code>test</code>	确保响应是 JPEG、GIF、PNG 或 WebP 格式的图像。否则, 将返回 415 (不支持的媒体类型) 错误。
<code>size</code>	<p>以 JSON 格式输出图像信息, 例如:</p> <pre>{ "img" : { "width": 100, "height": 100, "type": "gif" } }</pre> <p>出现错误时, 输出如下:</p> <pre>{ }</pre>
<code>rotate 90   180   270</code>	将图像逆时针旋转指定的度数。参数值可以包含变量。此模式可以单独使用, 或与 <code>resize</code> 和 <code>crop</code> 转换一起使用。
<code>resize width height</code>	按比例将图像缩小到指定的尺寸。若仅需缩小一个维度, 另一个维度可以指定为“-”。出现错误时, 服务器将返回 415 (不支持的媒体类型) 代码。参数值可以包含变量。当与 <code>rotate</code> 参数一起使用时, 旋转发生在缩小之后。
<code>crop width height</code>	按比例将图像缩小到较大一边的尺寸, 并通过另一边裁剪多余边缘。若仅需缩小一个维度, 另一个维度可以指定为“-”。出现错误时, 服务器将返回 415 (不支持的媒体类型) 代码。参数值可以包含变量。当与 <code>rotate</code> 参数一起使用时, 旋转发生在缩小 ** 之前 **。

### image\_filter\_buffer

语法	<code>image_filter_buffer size;</code>
默认值	<code>image_filter_buffer 1M;</code>
上下文	http, server, location

设置用于读取图像的缓冲区的最大大小。当大小超出时, 服务器返回 415 (不支持的媒体类型) 错误。

### image\_filter\_interlace

语法	<code>image_filter_interlace on   off;</code>
默认值	<code>image_filter_interlace off;</code>
上下文	http, server, location

如果启用, 最终图像将被交错。对于 JPEG, 最终图像将是“渐进式 JPEG”格式。



### image\_filter\_jpeg\_quality

语法	<code>image_filter_jpeg_quality quality;</code>
默认值	<code>image_filter_jpeg_quality 75;</code>
上下文	http, server, location

设置转换后 JPEG 图像的期望质量。可接受的值范围是 1 到 100。较低的值通常意味着图像质量较低和传输的数据较少。最大推荐值是 95。参数值可以包含变量。

### image\_filter\_sharpen

语法	<code>image_filter_sharpen percent;</code>
默认值	<code>image_filter_sharpen 0;</code>
上下文	http, server, location

增加最终图像的锐度。锐度百分比可以超过 100。0 值表示禁用锐化。参数值可以包含变量。

### image\_filter\_transparency

语法	<code>image_filter_transparency on   off;</code>
默认值	<code>image_filter_transparency on;</code>
上下文	http, server, location

定义在转换 GIF 图像或使用调色板指定颜色的 PNG 图像时是否保留透明度。失去透明度会导致图像质量更好。PNG 中的 alpha 通道透明度始终保留。

### image\_filter\_webp\_quality

语法	<code>image_filter_webp_quality quality;</code>
默认值	<code>image_filter_webp_quality 80;</code>
上下文	http, server, location

设置转换后 WebP 图像的期望质量。可接受的值范围是 1 到 100。较低的值通常意味着图像质量较低和传输的数据较少。参数值可以包含变量。

## 索引

该模块处理以斜杠字符 (/) 结尾的请求。这类请求也可以由 `http_autoindex` 和 `http_random_index` 模块处理。

## 配置示例

```
location / {
    index index.$geo.html index.html;
}
```

## 指令

### index

语法	<code>index file ...;</code>
默认值	<code>index index.html;</code>
上下文	http, server, location

定义将用作索引的文件。文件名可以包含变量。文件按照指定的顺序进行检查。列表的最后一个元素可以是一个具有绝对路径的文件。示例：

```
index index.$geo.html index.0.html /index.html;
```

需要注意的是，使用索引文件会导致内部重定向，并且请求可以在不同的位置进行处理。例如，使用以下配置：

```
location = / {
    index index.html;
}

location / {
    # ...
}
```

一个 "" 请求实际上将在第二个位置作为 `/index.html` 进行处理。

## JS

该模块用于在 njs 中实现处理程序——JavaScript 语言的一个子集。

在我们的代码库中, 该模块是 动态构建的, 并作为一个独立的软件包提供, 名为 `angie-module-njs` 或 `angie-pro-module-njs`。

### 配置示例

```
http {
    js_import http.js;

    js_set $foo      http.foo;
    js_set $summary  http.summary;
    js_set $hash     http.hash;

    resolver 127.0.0.53;

    server {
        listen 8000;

        location / {
            add_header X-Foo $foo;
            js_content http.baz;
        }

        location = /summary {
            return 200 $summary;
        }

        location = /hello {
            js_content http.hello;
        }

        location = /fetch {
            js_content          http.fetch;
            js_fetch_trusted_certificate /path/to/ISRG_Root_X1.pem;
        }

        location = /crypto {
            add_header Hash $hash;
            return 200;
        }
    }
}
```

http.js 文件:

```
function foo(r) {
  r.log("hello from foo() handler");
  return "foo";
}

function summary(r) {
  var a, s, h;

  s = "JS summary\n\n";

  s += "Method: " + r.method + "\n";
  s += "HTTP version: " + r.httpVersion + "\n";
  s += "Host: " + r.headersIn.host + "\n";
  s += "Remote Address: " + r.remoteAddress + "\n";
  s += "URI: " + r.uri + "\n";

  s += "Headers:\n";
  for (h in r.headersIn) {
    s += "  header '" + h + "' is '" + r.headersIn[h] + "'\n";
  }

  s += "Args:\n";
  for (a in r.args) {
    s += "  arg '" + a + "' is '" + r.args[a] + "'\n";
  }

  return s;
}

function baz(r) {
  r.status = 200;
  r.headersOut.foo = 1234;
  r.headersOut['Content-Type'] = "text/plain; charset=utf-8";
  r.headersOut['Content-Length'] = 15;
  r.sendHeader();
  r.send("nginx");
  r.send("java");
  r.send("script");

  r.finish();
}

function hello(r) {
  r.return(200, "Hello world!");
}

async function fetch(r) {
  let results = await Promise.all([ngx.fetch('https://google.com/'),
```

```
        ngx.fetch('https://google.ru/']));

    r.return(200, JSON.stringify(results, undefined, 4));
}

async function hash(r) {
    let hash = await crypto.subtle.digest('SHA-512', r.headersIn.host);
    r.setReturnValue(Buffer.from(hash).toString('hex'));
}

export default {foo, summary, baz, hello, fetch, hash};
```

## 指令

### js\_body\_filter

语法	<code>js_body_filter function   module.function [buffer_type=string   buffer];</code>
默认值	—
上下文	location, if in location, limit_except

设置一个 `njs` 函数作为响应体过滤器。过滤器函数为响应体的每个数据块调用，并具有以下参数：

<code>r</code>	HTTP 请求 对象
<code>data</code>	传入的数据块，可能是字符串或缓冲区，取决于 <code>buffer_type</code> 的值，默认是字符串。
<code>flags</code>	一个包含以下属性的对象： <code>last</code> — 一个布尔值 <code>true</code> — 如果数据是最后一个缓冲区。

过滤器函数可以通过调用 `r.sendBuffer()` 将其修改后的输入数据块传递给下一个过滤器。例如，要将响应体中的所有小写字母转换：

```
function filter(r, data, flags) {
    r.sendBuffer(data.toLowerCase(), flags);
}
```

要停止过滤（后续数据块将直接传递给客户端而不调用 `js_body_filter`），可以使用 `r.done()`。

如果过滤器函数更改了响应体的长度，则需要在 `js_header_filter` 中清除“Content-Length”响应头（如果有）以强制分块传输编码。

#### 备注

由于 `js_body_filter` 处理程序立即返回其结果, 仅支持同步操作。因此, 不支持异步操作, 如 `r.subrequest()` 或 `setTimeout()`。

### js\_content

语法	<code>js_content function   module.function;</code>
默认值	—
上下文	location, if in location, limit_except

设置一个 `njs` 函数作为位置内容处理程序。可以引用模块函数。

### js\_fetch\_buffer\_size

语法	<code>js_fetch_buffer_size size;</code>
默认值	<code>js_fetch_buffer_size 16k;</code>
上下文	http, server, location

设置用于通过 Fetch API 读取和写入的缓冲区大小。

### js\_fetch\_ciphers

语法	<code>js_fetch_ciphers ciphers;</code>
默认值	<code>js_fetch_ciphers HIGH:!aNULL:!MD5;</code>
上下文	http, server, location

指定用于通过 Fetch API 的 HTTPS 连接的启用密码。密码以 OpenSSL 库能够理解的格式指定。

可以使用“`openssl ciphers`”命令查看完整列表。

### js\_fetch\_max\_response\_buffer\_size

语法	<code>js_fetch_max_response_buffer_size size;</code>
默认值	<code>js_fetch_max_response_buffer_size 1m;</code>
上下文	http, server, location

设置通过 Fetch API 接收的响应的最大大小。

### js\_fetch\_protocols

语法	<code>js_fetch_protocols [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];</code>
默认值	<code>js_fetch_protocols TLSv1 TLSv1.1 TLSv1.2;</code>
上下文	<code>http, server, location</code>

启用通过 Fetch API 的 HTTPS 连接的指定协议。

### js\_fetch\_timeout

语法	<code>js_fetch_timeout <i>time</i>;</code>
默认值	<code>js_fetch_timeout 60s;</code>
上下文	<code>http, server, location</code>

定义 Fetch API 读取和写入的超时时间。超时时间仅在两次连续的读/写操作之间设置，而不是针对整个响应。如果在此时间内没有数据传输，连接将关闭。

### js\_fetch\_trusted\_certificate

语法	<code>js_fetch_trusted_certificate <i>file</i>;</code>
默认值	—
上下文	<code>http, server, location</code>

指定一个包含 CA 证书的 PEM 格式文件，用于通过 Fetch API 验证 HTTPS 证书。

### js\_fetch\_verify

语法	<code>js_fetch_verify on   off;</code>
默认值	<code>js_fetch_verify on;</code>
上下文	<code>http, server, location</code>

启用或禁用通过 Fetch API 验证 HTTPS 服务器证书。

### js\_fetch\_verify\_depth

语法	<code>js_fetch_verify_depth number;</code>
默认值	<code>js_fetch_verify_depth 100;</code>
上下文	http, server, location

设置通过 Fetch API 的 HTTPS 服务器证书链验证深度。

### js\_header\_filter

语法	<code>js_header_filter function   module.function;</code>
默认值	—
上下文	location, if in location, limit_except

设置一个 njs 函数作为响应头过滤器。该指令允许更改响应头的任意字段。

#### **i** 备注

由于 `js_header_filter` 处理程序立即返回其结果, 仅支持同步操作。因此, 不支持异步操作, 如 `r.subrequest()` 或 `setTimeout()`。

### js\_import

语法	<code>js_import module.js   export_name from module.js;</code>
默认值	—
上下文	http, server, location

导入一个在 njs 中实现位置和变量处理程序的模块。 `export_name` 用作命名空间来访问模块函数。如果未指定 `export_name`, 则使用模块名称作为命名空间。

```
js_import http.js;
```

在这里, 模块名称 `http` 用作访问导出的命名空间。如果导入的模块导出 `foo()`, 则使用 `http.foo` 来引用它。

可以指定多个 `js_import` 指令。



### js\_path

语法	<code>js_path path;</code>
默认值	—
上下文	http, server, location

设置 `njs` 模块的附加路径。

### js\_preload\_object

语法	<code>js_preload_object name.json   name from file.json;</code>
默认值	—
上下文	http, server, location

在配置时预加载一个不可变对象。`name` 用作全局变量的名称, 通过该变量可以在 `njs` 代码中访问对象。如果未指定 `name`, 则使用文件名作为变量名。

```
js_preload_object map.json;
```

在此示例中, `map` 被用作访问预加载对象时的名称。

可以指定多个 `js_preload_object` 指令。

### js\_set

语法	<code>js_set \$variable function   module.function;</code>
默认值	—
上下文	http, server, location

为指定变量设置 `njs` 函数。可以引用模块函数。

该函数在请求中首次引用变量时调用。具体时间取决于引用变量的阶段。这可以用于执行一些与变量评估无关的逻辑。例如, 如果变量仅在 `log_format` 指令中引用, 则其处理程序将在日志阶段才执行。此处理程序可用于在请求释放前进行一些清理工作。

#### 备注

由于 `js_set` 处理程序立即返回其结果, 因此仅支持同步回调。因此, 不支持诸如 `r.subrequest()` 或 `setTimeout()` 的异步回调。

## js\_shared\_dict\_zone

语法	<code>js_shared_dict_zone zone=name:size [timeout=time] [type=string   number] [evict];</code>
默认值	—
上下文	http

设置共享内存区域的名称和大小, 该区域在工作进程之间共享键值字典。

type	可选参数, 允许将值类型重新定义为 <code>number</code> , 默认情况下, 共享字典使用 <code>string</code> 作为键和值。
timeout	可选参数, 设置共享字典条目从区域中移除的时间。
evict	可选参数, 当区域存储耗尽时, 移除最旧的键值对。

示例:

```
example.conf:
# 创建一个具有字符串值的 1Mb 字典,
# 在 60 秒不活动后移除键值对:
js_shared_dict_zone zone=foo:1M timeout=60s;

# 创建一个具有字符串值的 512Kb 字典,
# 当区域耗尽时强制移除最旧的键值对:
js_shared_dict_zone zone=bar:512K timeout=30s evict;

# 创建一个具有数字值的 32Kb 永久字典:
js_shared_dict_zone zone=num:32k type=number;

example.js:
function get(r) {
    r.return(200, ngx.shared.foo.get(r.args.key));
}

function set(r) {
    r.return(200, ngx.shared.foo.set(r.args.key, r.args.value));
}

function delete(r) {
    r.return(200, ngx.shared.bar.delete(r.args.key));
}

function increment(r) {
    r.return(200, ngx.shared.num.incr(r.args.key, 2));
}
```

## js\_var

语法	<code>js_var \$variable [value];</code>
默认值	—
上下文	stream, server

声明一个 **可写** 变量。值可以包含文本、变量及其组合。与通过 `set` 指令创建的变量不同, 该变量在重定向后不会被覆盖。

## 请求参数

每个 HTTP njs 处理程序接收一个参数, 即 `request` 对象。

## 连接限制

该模块用于根据定义的键限制连接数量, 特别是限制来自单个 IP 地址的连接数量。

并不是所有的连接都会被计数。只有当连接请求正在被服务器处理且整个请求头已经被读取时, 该连接才会被计数。

## 配置示例

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        location /download/ {
            limit_conn addr 1;
        }
    }
}
```

## 指令

### limit\_conn

语法	<code>limit_conn zone number;</code>
默认值	—
上下文	http, server, location

设置共享内存区域和给定键值允许的最大连接数。当超过此限制时, 服务器将返回错误 作为请求的响应。例如, 以下指令

```
limit_conn_zone $binary_remote_addr zone=addr:10m;

server {
    location /download/ {
        limit_conn addr 1;
    }
}
```

允许每次一个 IP 地址只有一个连接。

#### **i** 备注

在 HTTP/2 和 SPDY 中, 每个并发请求被视为一个单独的连接。

可以有多个 `limit_conn` 指令。例如, 以下配置将限制每个客户端 IP 到服务器的连接数, 同时限制到虚拟服务器的总连接数:

```
limit_conn_zone $binary_remote_addr zone=perip:10m;
limit_conn_zone $server_name zone=perserver:10m;

server {
    ...
    limit_conn perip 10;
    limit_conn perserver 100;
}
```

这些指令从先前的配置级别继承, 前提是当前级别没有定义 `limit_conn` 指令。

### limit\_conn\_dry\_run

语法	<code>limit_conn_dry_run on   off;</code>
默认值	<code>limit_conn_dry_run off;</code>
上下文	http, server, location

启用试运行模式。在此模式下, 连接数量不受限制, 但在共享内存区域中, 过多的连接数量仍被正常计数。

### limit\_conn\_log\_level

语法	<code>limit_conn_log_level info   notice   warn   error;</code>
默认值	<code>limit_conn_log_level error;</code>
上下文	http, server, location

设置当服务器限制连接数量时所需的日志记录级别。

### limit\_conn\_status

语法	<code>limit_conn_status code;</code>
默认值	<code>limit_conn_status 503;</code>
上下文	http, server, location

设置对被拒绝请求返回的状态码。

### limit\_conn\_zone

语法	<code>limit_conn_zone key zone = name:size;</code>
默认值	—
上下文	http

为共享内存区域设置参数, 该区域将保存各种键的状态。特别是, 状态包括当前连接数。键可以包含文本、变量及其组合。空键值的请求不计数。

使用示例:

```
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

在这里, 客户端 IP 地址作为键。注意, 这里使用的是 `$binary_remote_addr` 变量, 而不是 `$remote_addr`。

`$remote_addr` 变量的大小可以从 7 到 15 字节变化。在 32 位平台上, 存储的状态占用 32 或 64 字节内存, 在 64 位平台上始终占用 64 字节。

`$binary_remote_addr` 变量的大小对于 IPv4 地址始终为 4 字节, 对于 IPv6 地址为 16 字节。在 32 位平台上, 存储的状态始终占用 32 或 64 字节内存, 在 64 位平台上占用 64 字节。

一个兆字节的区域可以保存大约 32,000 个 32 字节状态或大约 16,000 个 64 字节状态。如果区域存储耗尽, 服务器将对所有后续请求返回错误。

## 内置变量

`$limit_conn_status`

保存限制连接数的结果: PASSED、REJECTED 或 REJECTED\_DRY\_RUN

## 请求限制

该模块用于根据定义的键限制请求处理速率, 特别是限制来自单个 IP 地址的请求处理速率。限制是通过“漏桶”方法实现的。

## 配置示例

```
http {
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    ...

    server {

        ...

        location /search/ {
            limit_req zone=one burst=5;
        }
    }
}
```

## 指令

### limit\_req

语法	<code>limit_req zone number [burst=number] [nodelay   delay=number];</code>
默认值	—
上下文	http, server, location

设置共享内存区域和请求的最大突发大小。如果请求速率超过为某个区域配置的速率，它们的处理将被延迟，以便请求以定义的速率进行处理。多余的请求将被延迟，直到其数量超过最大突发大小，在这种情况下，请求将以错误终止。默认情况下，最大突发大小为零。例如，指令

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

server {
    location /search/ {
        limit_req zone=one burst=5;
    }
}
```

平均允许每秒最多 1 个请求，突发不超过 5 个请求。

如果不希望在请求被限制时延迟多余的请求，应使用参数 `nodelay`：

```
limit_req zone=one burst=5 nodelay;
```

参数 `delay` 指定延迟多余请求的限制。默认值为零，即所有多余请求都被延迟。

可以有几个 `limit_req` 指令。例如，以下配置将限制来自单个 IP 地址的请求处理速率，同时限制虚拟服务器的请求处理速率：

```
limit_req_zone $binary_remote_addr zone=perip:10m rate=1r/s;
limit_req_zone $server_name zone=perserver:10m rate=10r/s;

server {
    ...
    limit_req zone=perip burst=5 nodelay;
    limit_req zone=perserver burst=10;
}
```

这些指令仅在当前级别未定义 `limit_req` 指令时从上一级配置继承。

### limit\_req\_dry\_run

语法	<code>limit_req_dry_run on   off;</code>
默认值	<code>limit_req_dry_run off;</code>
上下文	http, server, location

启用试运行模式。在这种模式下, 请求处理速率不受限制, 但在共享内存区域中, 多余请求的数量仍按常规计数。

### limit\_req\_log\_level

语法	<code>limit_req_log_level info   notice   warn   error;</code>
默认值	<code>limit_req_log_level error;</code>
上下文	http, server, location

设置服务器因速率过高拒绝处理请求或延迟请求处理时的日志记录级别。延迟的日志记录级别比拒绝的低一个级别; 例如, 如果指定 `limit_req_log_level notice`, 则延迟以 `info` 级别记录。

### limit\_req\_status

语法	<code>limit_req_status code;</code>
默认值	<code>limit_req_status 503;</code>
上下文	http, server, location

设置响应被拒请求时返回的状态码。

### limit\_req\_zone

语法	<code>limit_req_zone key zone=name:size rate=rate;</code>
默认值	—
上下文	http

为共享内存区域设置参数, 该区域将保留各种键的状态。特别是, 状态存储当前多余请求的数量。键可以包含文本、变量及其组合。具有空键值的请求不计算在内。

使用示例:

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
```



在这里, 状态保存在一个 10 兆字节的区域 `one` 中, 此区域的平均请求处理速率不能超过每秒 1 个请求。

客户端 IP 地址用作键。请注意, 这里使用的是 `$binary_remote_addr` 变量, 而不是 `$remote_addr`。

`$binary_remote_addr` 变量的大小对于 IPv4 地址始终为 4 字节, 对于 IPv6 地址为 16 字节。存储的状态在 32 位平台上始终占用 64 字节, 在 64 位平台上占用 128 字节。

一个兆字节的区域可以保留大约 16,000 个 64 字节状态或大约 8,000 个 128 字节状态。

如果区域存储已满, 最近最少使用的状态将被移除。如果即便如此也无法创建新状态, 请求将以错误终止。

`rate` 以每秒请求数 (r/s) 指定。如果希望小于每秒一个请求的速率, 则以每分钟请求数 (r/m) 指定。例如, 半个请求每秒为 30r/m。

## 内置变量

### `$limit_req_status`

保持请求处理速率限制的结果: `PASSED`、`DELAYED`、`REJECTED`、`DELAYED_DRY_RUN` 或 `REJECTED_DRY_RUN`

日志 `###`

该模块以指定格式记录请求日志。

请求在处理结束的 `location` 上下文中记录。如果请求处理过程中发生 *internal redirect*, 则可能与原始 `location` 不同。

## 配置示例

```
log_format compression '$remote_addr - $remote_user [$time_local] '
                        '$request' $status $bytes_sent '
                        '$http_referer' '$http_user_agent' '$gzip_ratio';

access_log /spool/logs/angie-access.log compression buffer=32k;
```

## 指令

### `access_log`

语法	<code>access_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]];</code> <code>access_log off;</code>
默认	<code>access_log logs/access.log combined;</code> (路径取决于 <code>--http-log-path</code> 构建选项)
上下文	<code>http</code> , <code>server</code> , <code>location</code> , <code>if in location</code> , <code>limit_except</code>

设置缓冲日志写入的 `path`、`format` 和配置。可以在同一级别指定多个日志。通过在第一个参数中指定 “`syslog:`” 前缀可以配置记录到 `syslog`。特殊值 `off` 取消当前级别的所有 `access_log` 指令。如果未指定格式, 则使用预定义的 “`combined`” 格式。

如果使用了 `buffer` 或 `gzip` 参数, 则日志写入将被缓冲。

#### ⚠ 小心

缓冲区大小不得超过磁盘文件原子写入的大小。对于 FreeBSD, 此大小不受限制。

启用缓冲时, 数据将在以下情况下写入文件:

- 如果下一行日志无法放入缓冲区;
- 如果缓冲数据的时间超过了 `flush` 参数指定的时间;
- 当工作进程重新打开日志文件 或关闭时。

如果使用了 `gzip` 参数, 缓冲数据将在写入文件前压缩。可以将压缩级别设置在 1 (最快, 压缩率最低) 和 9 (最慢, 压缩率最高) 之间。默认情况下, 缓冲区大小为 64K 字节, 压缩级别设置为 1。由于数据是以原子块压缩的, 日志文件可以随时通过 “`zcat`” 解压或读取。

示例:

```
access_log /path/to/log.gz combined gzip flush=5m;
```

#### 🛑 重要

为了使 `gzip` 压缩生效, Angie 必须使用 `zlib` 库构建。

文件路径可以包含变量, 但此类日志具有一些限制:

- 使用工作进程凭证的 `user` 应具有在包含此类日志的目录中创建文件的权限;
- 缓冲写入不起作用;
- 每次日志写入时都会打开和关闭文件。然而, 由于频繁使用的文件的描述符可以存储在缓存中, 可以在 `open_log_file_cache` 指令的 `valid` 参数指定的时间内继续写入旧文件;
- 每次日志写入时都会检查请求根目录的存在性, 如果不存在则不创建日志。因此, 最好在同一级别指定 `root` 和 `access_log`:

```
server {  
    root          /spool/vhost/data/$host;  
    access_log    /spool/vhost/logs/$host;  
    ...  
}
```

`if` 参数启用条件日志记录。如果条件评估为 “0” 或空字符串, 请求将不会被记录。在以下示例中, 响应代码为 2xx 和 3xx 的请求将不会被记录:

```
map $status $loggable {
    ~^[23] 0;
    default 1;
}

access_log /path/to/access.log combined if=$loggable;
```

### log\_format

语法	<code>log_format name [escape=default   json   none] string ...;</code>
默认	<code>log_format combined "...";</code>
上下文	http

指定日志格式。

`escape` 参数允许设置变量的 `json` 或 `default` 字符转义, 默认使用 `default` 转义。值 `none` 禁用转义。

对于 `default` 转义, 字符”””, ”\” 和其它值小于 32 或大于 126 的字符被转义为”\xXX”。如果未找到变量值, 将记录一个连字符”-”。

对于 `json` 转义, 所有不允许在 JSON 字符串中的字符将被转义: 字符”””, ”\” 被转义为”\””” 和”\\”, 值小于 32 的字符被转义为”\n”, ”\r”, ”\t”, ”\b”, ”\f” 或”\u00XX”。

发送给客户端的报头行前缀为 `sent_http_`, 例如, `$sent_http_content_range`。

配置始终包含预定义的 `combined` 格式:

```
log_format combined '$remote_addr - $remote_user [$time_local] '
    '$request' $status $body_bytes_sent '
    '$http_referer' '$http_user_agent';
```

### open\_log\_file\_cache

语法	<code>open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time];</code> <code>open_log_file_cache off;</code>
默认	<code>open_log_file_cache off;</code>
上下文	http, server, location

定义一个缓存, 用于存储名称中包含变量的常用日志的文件描述符。该指令具有以下参数:

max	设置缓存中描述符的最大数量; 如果缓存已满, 则最近最少使用的 (LRU) 描述符将被关闭
inactive	设置缓存描述符在多长时间无访问后关闭的时间; 默认情况下为 10 秒
min_uses	设置在 <i>inactive</i> 参数定义的时间内文件使用次数的最小值, 以便描述符在缓存中保持打开状态; 默认情况下为 1
valid	设置检查文件是否仍然存在且名称相同的时间间隔; 默认情况下为 60 秒
off	禁用缓存

使用示例:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```

映射 ###

该模块创建的变量, 其值依赖于其他变量的值。

### 配置示例

```
map $http_host $name {
    hostnames;

    default      0;

    example.com  1;
    *.example.com 1;
    example.org  2;
    *.example.org 2;
    .example.net 3;
    wap.*        4;
}

map $http_user_agent $mobile {
    default      0;
    "~Opera Mini" 1;
}
```

### 指令

#### map

语法	map <i>string</i> <i>\$variable</i> { ... }
默认值	—
上下文	http

创建一个新变量。其值取决于第一个参数，该参数以包含变量的字符串形式指定，例如：

```
set $var1 "foo";
set $var2 "bar";

map $var1$var2 $new_variable {
    default "foobar_value";
}
```

在这里，变量 `$new_variable` 的值将由变量 `$var1` 和 `$var2` 组合而成，或者在这些变量未定义时使用默认值。

#### **i** 备注

由于变量仅在使用时才会被评估，即使声明了大量的“map”变量，也不会给请求处理增加额外的开销。

`map` 块内的参数指定了源值和结果值之间的映射关系。

源值可以指定为字符串或正则表达式。

字符串匹配时忽略大小写。

正则表达式要么以“~”符号开头表示区分大小写的匹配，要么以“~\*”符号开头表示不区分大小写的匹配。正则表达式可以包含命名和位置捕获，这些捕获随后可以与结果变量一起用于其他指令。

如果源值与下述特殊参数的名字之一匹配，则应在其前面加上“”符号。

结果值可以包含文本、变量及其组合。

还支持以下特殊参数：

<code>default value</code>	如果源值与未指定的变体都不匹配，则设置结果值。当未指定 <code>default</code> 时，默认结果值将是空字符串。
<code>hostnames</code>	表示源值可以是带有前缀或后缀掩码的主机名。此参数应在值列表之前指定。

例如，

```
*.example.com 1;
example.*     1;
```

以下两个记录

```
example.com 1;
*.example.com 1;
```

可以合并为：

```
.example.com 1;
```

<code>include file</code>	包含一个包含值的文件。可以有多个包含。
<code>volatile</code>	表示变量不可缓存。

如果源值与多个指定的变体匹配，例如同时匹配掩码和正则表达式，则选择第一个匹配的变体，优先顺序如下：

1. 不带掩码的字符串值
2. 带前缀掩码的最长字符串值，例如“\*.example.com”
3. 带后缀掩码的最长字符串值，例如“mail.\*”
4. 第一个匹配的正则表达式（按配置文件中的出现顺序）
5. 默认值 (*default*)

### map\_hash\_bucket\_size

语法	<code>map_hash_bucket_size size;</code>
默认值	<code>map_hash_bucket_size 32 64 128;</code>
上下文	http

设置 `map` 变量哈希表的桶大小。默认值取决于处理器的缓存行大小。设置哈希表的详细信息请参见 单独配置。

### map\_hash\_max\_size

语法	<code>map_hash_max_size size;</code>
默认值	<code>map_hash_max_size 2048;</code>
上下文	http

设置 `map` 变量哈希表的最大大小。设置哈希表的详细信息请参见 单独配置。

## Memcached

该模块用于从 memcached 服务器获取响应。键在 `$memcached_key` 变量中设置。响应应该通过外部手段预先放入 memcached 中。

### 配置示例

```
server {
    location / {
        set             $memcached_key "$uri?$args";
        memcached_pass  host:11211;
        error_page      404 502 504 = @fallback;
    }

    location @fallback {
        proxy_pass      http://backend;
    }
}
```

## 指令

### memcached\_bind

语法	<code>memcached_bind address [transparent]   off;</code>
默认值	—
上下文	http, server, location

使到 memcached 服务器的外发连接从指定的本地 IP 地址发起, 带有可选的端口。参数值可以包含变量。特殊值 `off` 取消从上一级配置继承的 `memcached_bind` 指令的效果, 这允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 memcached 服务器的外发连接从非本地 IP 地址发起, 例如, 从客户端的真实 IP 地址发起:

```
memcached_bind $remote_addr transparent;
```

为了使此参数生效, 通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上, 如果指定了 `transparent` 参数, 则不需要此操作, 因为工作进程从主进程继承 `CAP_NET_RAW` 能力。

#### 重要

必须配置内核路由表以截获来自 memcached 服务器的网络流量。

### memcached\_buffer\_size

语法	<code>memcached_buffer_size size;</code>
默认值	<code>memcached_buffer_size 4k 8k;</code>
上下文	<code>http, server, location</code>

设置用于读取从 memcached 服务器接收到的响应第一部分的缓冲区大小。响应一接收到就同步传递给客户端。

### memcached\_connect\_timeout

语法	<code>memcached_connect_timeout time;</code>
默认值	<code>memcached_connect_timeout 60s;</code>
上下文	<code>http, server, location</code>

定义与 memcached 服务器建立连接的超时时间。需要注意的是, 此超时时间通常不能超过 75 秒。

### memcached\_gzip\_flag

语法	<code>memcached_gzip_flag flag;</code>
默认值	—
上下文	<code>http, server, location</code>

启用测试 memcached 服务器响应中的标志, 并在标志设置时将响应头的“Content-Encoding”字段设置为“gzip”。

### memcached\_next\_upstream

语法	<code>memcached_next_upstream error   timeout   invalid_response   not_found   off ...;</code>
默认值	<code>memcached_next_upstream error timeout;</code>
上下文	<code>http, server, location</code>

指定在何种情况下请求应传递给上游池中的下一个服务器:



error	在与服务器建立连接、传递请求或读取响应头时发生错误;
timeout	在与服务器建立连接、传递请求或读取响应头时发生超时;
invalid_response	服务器返回空或无效响应;
not_found	在服务器上未找到响应;
off	禁止请求传递给下一个服务器。

**i 备注**

应记住, 只有在尚未向客户端发送任何内容时, 才能将请求传递给下一个服务器。也就是说, 如果在响应传输过程中发生错误或超时, 则无法解决。

该指令还定义了与服务器通信的不成功尝试 的标准。

error	timeout	总是被视为不成功尝试, 即使它们未在指令中指定
invalid_response		
not_found		从不被视为不成功尝试

将请求传递给下一个服务器的尝试次数可以通过尝试次数 和时间 进行限制。

**memcached\_next\_upstream\_timeout**

语法	memcached_next_upstream_timeout <i>time</i> ;
默认值	memcached_next_upstream_timeout 0;
上下文	http, server, location

限制请求可以传递到下一个 服务器的时间。

0	关闭此限制
---	-------

**memcached\_next\_upstream\_tries**

语法	memcached_next_upstream_tries <i>number</i> ;
默认值	memcached_next_upstream_tries 0;
上下文	http, server, location

限制请求传递到下一个 服务器的尝试次数。

0	关闭此限制
---	-------

### memcached\_pass

语法	<code>memcached_pass uri;</code>
默认值	—
上下文	location, if in location

设置 memcached 服务器地址。地址可以指定为域名或 IP 地址, 以及端口:

```
memcached_pass localhost:11211;
```

或者指定为 UNIX 域套接字路径:

```
memcached_pass unix:/tmp/memcached.socket;
```

如果一个域名解析为多个地址, 它们将以轮循方式使用。此外, 地址可以指定为服务器组。如果使用组, 则不能为其指定端口; 相反, 应为组内的每个服务器分别指定端口。

### memcached\_read\_timeout

语法	<code>memcached_read_timeout time;</code>
默认值	<code>memcached_read_timeout 60s;</code>
上下文	http, server, location

定义从 memcached 服务器读取响应的超时时间。超时时间仅在两次连续读取操作之间设置, 而不是用于整个响应的传输。如果 memcached 服务器在此时间内没有传输任何内容, 则连接将关闭。

### memcached\_send\_timeout

语法	<code>memcached_send_timeout time;</code>
默认值	<code>memcached_send_timeout 60s;</code>
上下文	http, server, location

设置向 memcached 服务器传输请求的超时时间。超时时间仅在两次连续写入操作之间设置, 而不是用于整个请求的传输。如果 memcached 服务器在此时间内没有接收到任何内容, 则连接将关闭。

## memcached\_socket\_keepalive

语法	memcached_socket_keepalive on   off;
默认值	memcached_socket_keepalive off;
上下文	http, server, location

配置到 memcached 服务器的外发连接的“TCP keepalive”行为。

""	默认情况下, 操作系统的设置对套接字有效。
on	为套接字打开 <i>SO_KEEPALIVE</i> 套接字选项。

## 内置变量

`$memcached_key`

定义从 memcached 服务器获取响应的键。

## 镜像

该模块通过创建后台镜像子请求来实现对原始请求的镜像。对镜像子请求的响应将被忽略。

## 配置示例

```
location / {
    mirror /mirror;
    proxy_pass http://backend;
}

location = /mirror {
    internal;
    proxy_pass http://test_backend$request_uri;
}
```

## 指令

### mirror

语法	<code>mirror uri   off;</code>
默认值	<code>mirror off;</code>
上下文	http, server, location

设置原始请求将被镜像到的 URI。可以在同一级配置中指定多个镜像。

### mirror\_request\_body

语法	<code>mirror_request_body on   off;</code>
默认值	<code>mirror_request_body on;</code>
上下文	http, server, location

指示客户端请求体是否被镜像。当启用时, 客户端请求体将在创建镜像子请求之前被读取。在这种情况下, 由 `proxy_request_buffering`、`fastcgi_request_buffering`、`scgi_request_buffering` 和 `uwsgi_request_buffering` 指令设置的非缓冲客户端请求体代理将被禁用。

```
location / {
    mirror /mirror;
    mirror_request_body off;
    proxy_pass http://backend;
}

location = /mirror {
    internal;
    proxy_pass http://log_backend;
    proxy_pass_request_body off;
    proxy_set_header Content-Length "";
    proxy_set_header X-Original-URI $request_uri;
}
```

## MP4

该模块提供对 MP4 文件的伪流式传输服务器端支持。此类文件通常具有 .mp4、.m4v 或 .m4a 文件扩展名。

当从源码构建时 (请参阅 `sourcebuild`), 默认情况下不会构建此模块; 应使用 `--with-http_mp4_module` build option 启用。

在来自 `our repos` 的软件包和镜像中, 已包含此模块。

伪流式传输与兼容的媒体播放器协同工作。播放器通过 HTTP 请求向服务器发送指定开始时间的查询字符串参数（简单命名为 `start`，以秒为单位），服务器响应流的位置与请求时间相对应，例如：

```
http://example.com/elephants_dream.mp4?start=238.88
```

这允许在任何时间进行随机跳转，或者在时间轴中间开始播放。

为了支持跳转，基于 H.264 的格式在所谓的“moov atom”中存储元数据。它是文件的一部分，包含整个文件的索引信息。

要开始播放，播放器需要首先读取元数据。这是通过发送带有 `start=0` 参数的特殊请求完成的。许多编码软件将元数据插入文件的末尾。这对于伪流式传输来说并不理想，因为播放器必须先下载整个文件才能开始播放。如果元数据位于文件的开头，Angie 只需开始发送文件内容即可。如果元数据位于文件末尾，Angie 必须读取整个文件并准备一个新流，使元数据在媒体数据之前。这涉及一些 CPU、内存和磁盘 I/O 开销，因此最好提前 `prepare` 原始文件以支持伪流式传输，而不是让 Angie 在每次这样的请求中进行处理。

该模块还支持 HTTP 请求的 `end` 参数，该参数设置播放的终点。`end` 参数可以与 `start` 参数一起指定或单独指定：

```
http://example.com/elephants_dream.mp4?start=238.88&end=555.55
```

对于具有非零 `start` 或 `end` 参数的匹配请求，Angie 将从文件中读取元数据，准备具有请求时间范围的流，并将其发送给客户端。这与上述描述的开销相同。

如果 `start` 参数指向非关键视频帧，则此类视频的开头将被破坏。为解决此问题，视频可以在 `start` 点之前和它们之间的所有中间帧之前插入关键帧。这些帧将通过编辑列表从播放中隐藏。

如果匹配请求不包含 `start` 和 `end` 参数，则没有开销，文件仅作为静态资源发送。一些播放器还支持字节范围请求，因此不需要此模块。

#### 小心

如果之前使用了第三方 mp4 模块，应将其禁用。

类似的 FLV 文件伪流式传输支持由 *FLV* 提供。

#### 配置示例

```
set_real_ip_from 192.168.1.0/24;  
set_real_ip_from 192.168.2.1;  
set_real_ip_from 2001:0db8::/32;  
real_ip_header X-Forwarded-For;  
real_ip_recursive on;
```

## 指令

### mp4

Syntax	mp4;
默认值	—
Context	location

在周围位置启用模块处理。

### mp4\_buffer\_size

Syntax	mp4_buffer_size <i>size</i> ;
默认值	mp4_buffer_size 512K;
Context	http, server, location

设置用于处理 MP4 文件的缓冲区的初始大小。

### mp4\_max\_buffer\_size

Syntax	mp4_max_buffer_size <i>size</i> ;
默认值	mp4_max_buffer_size 10M;
Context	http, server, location

在处理元数据期间, 可能需要更大的缓冲区。其大小不能超过指定大小, 否则 Angie 将返回 500 (内部服务器错误) 服务器错误, 并记录以下消息:

```
"/some/movie/file.mp4" mp4 moov atom is too large: 12583268, you may want to increase  
mp4_max_buffer_size
```

### mp4\_limit\_rate

Syntax	mp4_limit_rate on   off   <i>factor</i> ;
默认值	mp4_limit_rate off;
Context	http, server, location

限制将请求的 MP4 文件传输给客户端的速度。要计算限制, *factor* 乘以文件的平均比特率。

- off 值禁用速率限制。

- on 值设置 *factor* 为 1.1。
- 限制在达到 *mp4\_limit\_rate\_after* 设置的值后应用。

请求的速率限制是单独的：如果客户端打开两个连接，结果速率加倍。在这方面，请考虑使用 *limit\_conn* 和相关指令。

### mp4\_limit\_rate\_after

Syntax	<code>mp4_limit_rate_after time;</code>
默认值	<code>mp4_limit_rate_after 60s;</code>
Context	http, server, location

设置（在 playback time 中）传输的媒体数据量，以触发 *mp4\_limit\_rate* 设置的速率限制。

### mp4\_start\_key\_frame

Syntax	<code>mp4_start_key_frame on   off;</code>
默认值	<code>mp4_start_key_frame off;</code>
Context	http, server, location

强制输出视频始终以关键视频帧开始。如果 *start* 参数不指向关键帧，初始帧将使用 mp4 编辑列表隐藏。主要播放器和浏览器如 Chrome、Safari、QuickTime 和 ffmpeg 支持编辑列表，Firefox 部分支持。

## Perl

该模块用于在 Perl 中实现位置和变量处理程序，并将 Perl 调用插入到 SSI 中。

当从源代码 构建时，默认情况下不会构建此模块；应通过 配置选项启用 `--with-http_perl_module`。

在我们的存储库中，该模块是 动态构建的，并作为一个名为 `angie-module-perl` 或 `angie-pro-module-perl` 的独立包提供。

### 重要

此模块需要 Perl 版本 5.6.1 或更高版本。C 编译器应与用于构建 Perl 的编译器兼容。

## 已知问题

该模块是实验性的, 购买者需自担风险。

为了让 Perl 在重新配置期间重新编译修改过的模块, 它应该使用参数 `-Dusemultiplicity=yes` 或 `-Dusetthreads=yes` 构建。此外, 为了减少 Perl 在运行时的内存泄漏, 它应该使用参数 `-Dusemymalloc=no` 构建。要检查已经构建的 Perl 中这些参数的值 (示例中指定了首选值), 可以运行:

```
$ perl -V:usemultiplicity -V:usemymalloc | br |  
usemultiplicity='define'; | br |  
usemymalloc='n';
```

注意, 在使用新的 `-Dusemultiplicity=yes` 或 `-Dusetthreads=yes` 参数重新构建 Perl 后, 所有二进制 Perl 模块也都必须重新构建——它们将无法与新的 Perl 一起工作。

在每次重新配置后, 主进程和工作进程可能会变大。如果主进程增长到不可接受的大小, 可以在不更改可执行文件的情况下应用实时升级过程。

当 Perl 模块执行长时间运行的操作时, 例如解析域名、连接到另一个服务器或查询数据库, 分配给当前工作进程的其他请求将不会被处理。因此, 建议仅执行具有可预测且较短执行时间的操作, 例如访问本地文件系统。

## 配置示例

```
http {  
  
    perl_modules perl/lib;  
    perl_require hello.pm;  
  
    perl_set $msie6 '  
  
        sub {  
            my $r = shift;  
            my $ua = $r->header_in("User-Agent");  
  
            return "" if $ua =~ /Opera/;  
            return "1" if $ua =~ /MSIE [6-9]\.\d+/  
            return "";  
        }  
  
    ;  
  
    server {  
        location / {  
            perl hello::handler;  
        }  
    }  
}
```



`perl/lib/hello.pm` 模块:

```
package hello;

use nginx;

sub handler {
    my $r = shift;

    $r->send_http_header("text/html");
    return OK if $r->header_only;

    $r->print("hello!\n<br/>");

    if (-f $r->filename or -d _) {
        $r->print($r->uri, " exists!\n");
    }

    return OK;
}

1;
__END__
```

## 指令

### perl

语法	perl 模块:: 函数   'sub { ... }';
默认值	—
上下文	location, limit_except

为给定位置设置 Perl 处理程序。

### perl\_modules

语法	perl_modules 路径;
默认值	—
上下文	http

设置 Perl 模块的附加路径。

## perl\_require

语法	perl_require 模块;
默认值	—
上下文	http

定义将在每次重新配置时加载的模块名称。可以存在多个 `perl_require` 指令。

## perl\_set

语法	perl_set <i>\$variable</i> <i>module</i> :: <i>function</i>   'sub { ... }';
默认值	—
上下文	http

为指定变量安装 Perl 处理程序。

## 从 SSI 调用 Perl

调用 Perl 的 SSI 命令格式如下:

```
<!--# perl sub="module::function" arg="parameter1" arg="parameter2" ...  
-->
```

## `$r` 请求对象方法

`$r->args`

返回请求参数。

`$r->filename`

返回与请求 URI 对应的文件名。

`$r->has_request_body (handler)`

如果请求中没有主体, 则返回 0。如果有主体, 为请求设置指定的处理程序并返回 1。在读取请求主体后, Angie 将调用指定的处理程序。注意, 处理程序函数应通过引用传递。示例:

```
package hello;

use nginx;

sub handler {
    my $r = shift;

    if ($r->request_method ne "POST") {
        return DECLINED;
    }

    if ($r->has_request_body(\&post)) {
        return OK;
    }

    return HTTP_BAD_REQUEST;
}

sub post {
    my $r = shift;

    $r->send_http_header;

    $r->print("request_body: \"", $r->request_body, "\"<br/>");
    $r->print("request_body_file: \"", $r->request_body_file, "\"<br/>\n");

    return OK;
}

1;

__END__
```

```
$r->allow_ranges
```

在发送响应时启用字节范围的使用。

```
$r->discard_request_body
```

指示 Angie 丢弃请求主体。

```
$r->header_in (field)
```

返回指定客户端请求头字段的值。

```
$r->header_only
```

确定是发送整个响应还是仅发送其头部给客户端。

```
$r->header_out (field, value)
```

为指定的响应头字段设置值。

```
$r->internal_redirect (uri)
```

执行到指定 URI 的内部重定向。实际重定向在 Perl 处理程序执行完成后进行。方法接受转义的 URIs, 并支持重定向到命名位置。

```
$r->log_error (errno, message)
```

将指定的消息写入 *error\_log*。如果 *errno* 非零, 则会将错误代码及其描述附加到消息中。

```
$r->print (text, ...)
```

将数据传递给客户端。

```
$r->request_body
```

如果未将客户端请求主体写入临时文件, 则返回客户端请求主体。要确保客户端请求主体在内存中, 其大小应受限于 `client_max_body_size`, 并且应使用 `client_body_buffer_size` 设置足够的缓冲区大小。

```
$r->request_body_file
```

返回包含客户端请求主体的文件名。处理完成后, 应删除该文件。要始终将请求主体写入文件, 应启用 `client_body_in_file_only`。

```
$r->request_method
```

返回客户端请求的 HTTP 方法。

```
$r->remote_addr
```

返回客户端 IP 地址。

```
$r->flush
```

立即将数据发送给客户端。

```
$r->sendfile (name [, offset [, length ]])
```

将指定文件内容发送给客户端。可选参数指定要传输的数据的初始偏移量和长度。实际数据传输在 Perl 处理程序完成后进行。

```
$r->send_http_header ([type])
```

将响应头发送给客户端。可选的类型参数设置 “Content-Type” 响应头字段的值。如果值为空字符串, “Content-Type” 头字段将不会发送。

```
$r->status (code)
```

设置响应代码。

**`$r->sleep (milliseconds, handler)`**

设置指定的处理程序并停止请求处理指定的时间。同时, Angie 继续处理其他请求。在指定时间过去后, Angie 将调用安装的处理程序。注意, 处理程序函数应通过引用传递。为了在处理程序之间传递数据, 应使用 `$r->variable()`。示例:

```
package hello;

use nginx;

sub handler {
    my $r = shift;

    $r->discard_request_body;
    $r->variable("var", "OK");
    $r->sleep(1000, \&next);

    return OK;
}

sub next {
    my $r = shift;

    $r->send_http_header;
    $r->print($r->variable("var"));

    return OK;
}

1;

__END__
```

**`$r->unescape (text)`**

解码以 “%XX” 形式编码的文本。

```
$r->uri
```

返回请求 URL。

```
$r->variable (name [, value ])
```

返回或设置指定变量的值。变量对每个请求都是本地的。

## Prometheus

收集 Angie *metrics*, 使用配置定义的模板, 并以 Prometheus 格式返回由模板生成的指标。

### ⚠ 注意

要收集这些指标, 请在适当的上下文中使用以下指令启用共享内存区域:

- 在 `http_upstream` 或 `stream_upstream` 中使用 `zone` 指令;
- 使用 `status_zone` 指令;
- 在 `resolver` 指令中使用 `status_zone` 参数。

## 配置示例

三个指标收集服务器共享内存区域的请求统计信息, 合并到 `custom` 模板中, 并在 `/p8s` 暴露:

```
http {  
  
    prometheus_template custom {  
        'angie_http_server_zones_requests_total{zone="$1"}' $p8s_value  
        path=~^/http/server_zones/([~/]+)/requests/total$  
        type=counter;  
  
        'angie_http_server_zones_requests_processing{zone="$1"}' $p8s_value  
        path=~^/http/server_zones/([~/]+)/requests/processing$  
        type=gauge;  
  
        'angie_http_server_zones_requests_discarded{zone="$1"}' $p8s_value  
        path=~^/http/server_zones/([~/]+)/requests/discarded$  
        type=counter;  
    }  
  
    # ...  
  
    server {
```

```
listen 80;

location =/p8s {
    prometheus custom;
}

# ...

}
```

Angie 有一个补充的 `prometheus_all.conf` 文件, 定义了一些常用的指标, 用作 `all` 模板:

### 文件内容 (Angie)

```
prometheus_template all {

angie_connections_accepted $p8s_value
    path=/connections/accepted
    type=counter
    'help=The total number of accepted client connections.';

angie_connections_dropped $p8s_value
    path=/connections/dropped
    type=counter
    'help=The total number of dropped client connections.';

angie_connections_active $p8s_value
    path=/connections/active
    type=gauge
    'help=The current number of active client connections.';

angie_connections_idle $p8s_value
    path=/connections/idle
    type=gauge
    'help=The current number of idle client connections.';

'angie_slabs_pages_used{zone="$1"}' $p8s_value
    path=~^/slabs/([~/]+)/pages/used$
    type=gauge
    'help=The number of currently used memory pages in a slab zone.';

'angie_slabs_pages_free{zone="$1"}' $p8s_value
    path=~^/slabs/([~/]+)/pages/free$
    type=gauge
    'help=The number of currently free memory pages in a slab zone.';
```



```
'angie_slabs_pages_slots_used{zone="$1",size="$2"}' $p8s_value
    path=~~/slabs/([~/]+)/slots/([~/]+)/used$
    type=gauge
    'help=The number of currently used memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_free{zone="$1",size="$2"}' $p8s_value
    path=~~/slabs/([~/]+)/slots/([~/]+)/free$
    type=gauge
    'help=The number of currently free memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_reqs{zone="$1",size="$2"}' $p8s_value
    path=~~/slabs/([~/]+)/slots/([~/]+)/reqs$
    type=counter
    'help=The total number of attempts to allocate a memory slot of a specific size in a slab_
↪zone.';

'angie_slabs_pages_slots_fails{zone="$1",size="$2"}' $p8s_value
    path=~~/slabs/([~/]+)/slots/([~/]+)/fails$
    type=counter
    'help=The number of unsuccessful attempts to allocate a memory slot of a specific size in_
↪a slab zone.';

'angie_resolvers_queries{zone="$1",type="$2"}' $p8s_value
    path=~~/resolvers/([~/]+)/queries/([~/]+)$
    type=counter
    'help=The number of queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_sent{zone="$1",type="$2"}' $p8s_value
    path=~~/resolvers/([~/]+)/sent/([~/]+)$
    type=counter
    'help=The number of sent DNS queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_responses{zone="$1",status="$2"}' $p8s_value
    path=~~/resolvers/([~/]+)/responses/([~/]+)$
    type=counter
    'help=The number of resolution results with a specific status in a resolver zone.';

'angie_http_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
    path=~~/http/server_zones/([~/]+)/ssl/handshaked$
    type=counter
    'help=The total number of successful SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_reuses{zone="$1"}' $p8s_value
    path=~~/http/server_zones/([~/]+)/ssl/reuses$
```

```

type=counter
'help=The total number of session reuses during SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_timedout{zone="$1"}' $p8s_value
path=~^/http/server_zones/([^/]+)/ssl/timedout$
type=counter
'help=The total number of timed-out SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_failed{zone="$1"}' $p8s_value
path=~^/http/server_zones/([^/]+)/ssl/failed$
type=counter
'help=The total number of failed SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_requests_total{zone="$1"}' $p8s_value
path=~^/http/server_zones/([^/]+)/requests/total$
type=counter
'help=The total number of client requests received in an HTTP server zone.';

'angie_http_server_zones_requests_processing{zone="$1"}' $p8s_value
path=~^/http/server_zones/([^/]+)/requests/processing$
type=gauge
'help=The number of client requests currently being processed in an HTTP server zone.';

'angie_http_server_zones_requests_discarded{zone="$1"}' $p8s_value
path=~^/http/server_zones/([^/]+)/requests/discarded$
type=counter
'help=The total number of client requests completed in an HTTP server zone without sending
↪a response.';

'angie_http_server_zones_responses{zone="$1",code="$2"}' $p8s_value
path=~^/http/server_zones/([^/]+)/responses/([^/]+$
type=counter
'help=The number of responses with a specific status in an HTTP server zone.';

'angie_http_server_zones_data_received{zone="$1"}' $p8s_value
path=~^/http/server_zones/([^/]+)/data/received$
type=counter
'help=The total number of bytes received from clients in an HTTP server zone.';

'angie_http_server_zones_data_sent{zone="$1"}' $p8s_value
path=~^/http/server_zones/([^/]+)/data/sent$
type=counter
'help=The total number of bytes sent to clients in an HTTP server zone.';

```

```
'angie_http_location_zones_requests_total{zone="$1"}' $p8s_value
  path=~^/http/location_zones/([^/]+)/requests/total$
  type=counter
  'help=The total number of client requests in an HTTP location zone.';

'angie_http_location_zones_requests_discarded{zone="$1"}' $p8s_value
  path=~^/http/location_zones/([^/]+)/requests/discarded$
  type=counter
  'help=The total number of client requests completed in an HTTP location zone without
↔sending a response.';

'angie_http_location_zones_responses{zone="$1",code="$2"}' $p8s_value
  path=~^/http/location_zones/([^/]+)/responses/([^/]+)$
  type=counter
  'help=The number of responses with a specific status in an HTTP location zone.';

'angie_http_location_zones_data_received{zone="$1"}' $p8s_value
  path=~^/http/location_zones/([^/]+)/data/received$
  type=counter
  'help=The total number of bytes received from clients in an HTTP location zone.';

'angie_http_location_zones_data_sent{zone="$1"}' $p8s_value
  path=~^/http/location_zones/([^/]+)/data/sent$
  type=counter
  'help=The total number of bytes sent to clients in an HTTP location zone.';

'angie_http_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
  path=~^/http/upstreams/([^/]+)/peers/([^/]+)/state$
  type=gauge
  'help=The current state of an upstream peer in "HTTP": 1 - up, 2 - down, 3 - unavailable,
↔or 4 - recovering.';

'angie_http_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/current$
  type=gauge
  'help=The number of requests currently being processed by an upstream peer in "HTTP".';

'angie_http_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/total$
  type=counter
  'help=The total number of attempts to use an upstream peer in "HTTP".';

'angie_http_upstreams_peers_responses{upstream="$1",peer="$2",code="$3"}' $p8s_value
```

```
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/responses/([^/]+)$
type=counter
'help=The number of responses with a specific status received from an upstream peer in
↔"HTTP".';

'angie_http_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/sent$
type=counter
'help=The total number of bytes sent to an upstream peer in "HTTP".';

'angie_http_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/received$
type=counter
'help=The total number of bytes received from an upstream peer in "HTTP".';

'angie_http_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/fails$
type=counter
'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↔"HTTP".';

'angie_http_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/unavailable$
type=counter
'help=The number of times when an upstream peer in "HTTP" became "unavailable" due to
↔reaching the max_fails limit.';

'angie_http_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/downtime$
type=counter
'help=The total time (in milliseconds) that an upstream peer in "HTTP" was "unavailable".';

'angie_http_upstreams_keepalive{upstream="$1"}' $p8s_value
path=~^/http/upstreams/([^/]+)/keepalive$
type=gauge
'help=The number of currently cached keepalive connections for an HTTP upstream.';

'angie_http_caches_responses{zone="$1",status="$2"}' $p8s_value
path=~^/http/caches/([^/]+)/([^/]+)/responses$
type=counter
'help=The total number of responses processed in an HTTP cache zone with a specific cache
↔status.';

'angie_http_caches_bytes{zone="$1",status="$2"}' $p8s_value
```

```

path=~~/http/caches/([^/]+)/([^/]+)/bytes$
type=counter
'help=The total number of bytes processed in an HTTP cache zone with a specific cache_
↔status.';

'angie_http_caches_responses_written{zone="$1",status="$2"}' $p8s_value
path=~~/http/caches/([^/]+)/([^/]+)/responses_written$
type=counter
'help=The total number of responses written to an HTTP cache zone with a specific cache_
↔status.';

'angie_http_caches_bytes_written{zone="$1",status="$2"}' $p8s_value
path=~~/http/caches/([^/]+)/([^/]+)/bytes_written$
type=counter
'help=The total number of bytes written to an HTTP cache zone with a specific cache status.
↔';

'angie_http_caches_size{zone="$1"}' $p8s_value
path=~~/http/caches/([^/]+)/size$
type=gauge
'help=The current size (in bytes) of cached responses in an HTTP cache zone.';

'angie_http_caches_shards_size{zone="$1",path="$2"}' $p8s_value
path=~~/http/caches/([^/]+)/shards/([^/]+)/size$
type=gauge
'help=The current size (in bytes) of cached responses in a shard path of an HTTP cache_
↔zone.';

'angie_http_limit_conns{zone="$1",status="$2"}' $p8s_value
path=~~/http/limit_conns/([^/]+)/([^/]+)/$
type=counter
'help=The number of requests processed by an HTTP limit_conn zone with a specific result.';

'angie_http_limit_reqs{zone="$1",status="$2"}' $p8s_value
path=~~/http/limit_reqs/([^/]+)/([^/]+)/$
type=counter
'help=The number of requests processed by an HTTP limit_reqs zone with a specific result.';

'angie_stream_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([^/]+)/ssl/handshaked$
type=counter
'help=The total number of successful SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_reuses{zone="$1"}' $p8s_value

```

```

path=~~/stream/server_zones/([~/]+)/ssl/reuses$
type=counter
'help=The total number of session reuses during SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_timedout{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/ssl/timedout$
type=counter
'help=The total number of timed-out SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_failed{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/ssl/failed$
type=counter
'help=The total number of failed SSL handshakes in a stream server zone.';

'angie_stream_server_zones_connections_total{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/total$
type=counter
'help=The total number of client connections received in a stream server zone.';

'angie_stream_server_zones_connections_processing{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/processing$
type=gauge
'help=The number of client connections currently being processed in a stream server zone.';

'angie_stream_server_zones_connections_discarded{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/discarded$
type=counter
'help=The total number of client connections completed in a stream server zone without
↳ establishing a session.';

'angie_stream_server_zones_connections_passed{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/passed$
type=counter
'help=The total number of client connections in a stream server zone passed for handling
↳ to a different listening socket.';

'angie_stream_server_zones_sessions{zone="$1",status="$2"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/sessions/([~/]+)/$
type=counter
'help=The number of sessions finished with a specific status in a stream server zone.';

'angie_stream_server_zones_data_received{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/data/received$
type=counter
'help=The total number of bytes received from clients in a stream server zone.';

```

```
'angie_stream_server_zones_data_sent{zone="$1"}' $p8s_value
  path=~~/stream/server_zones/([~/]+)/data/sent$
  type=counter
  'help=The total number of bytes sent to clients in a stream server zone.';

'angie_stream_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/state$
  type=gauge
  'help=The current state of an upstream peer in "stream": 1 - up, 2 - down, 3 - unavailable,
↔ or 4 - recovering.';

'angie_stream_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/selected/current$
  type=gauge
  'help=The number of sessions currently being processed by an upstream peer in "stream".';

'angie_stream_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/selected/total$
  type=counter
  'help=The total number of attempts to use an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/data/sent$
  type=counter
  'help=The total number of bytes sent to an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/data/received$
  type=counter
  'help=The total number of bytes received from an upstream peer in "stream".';

'angie_stream_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/health/fails$
  type=counter
  'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↔"stream".';

'angie_stream_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/health/unavailable$
  type=counter
  'help=The number of times when an upstream peer in "stream" became "unavailable" due to
↔reaching the max_fails limit.';
```

```
'angie_stream_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([^/]+)/peers/([^/]+)/health/downtime$
  type=counter
  'help=The total time (in milliseconds) that an upstream peer in "stream" was "unavailable".
↪';
}

map $p8s_value $p8st_all_ups_state {
  volatile;
  "up"          1;
  "down"        2;
  "unavailable" 3;
  "recovering"  4;
  default       0;
}
```

### 文件内容 (Angie PRO)

```
prometheus_template all {

angie_connections_accepted $p8s_value
  path=/connections/accepted
  type=counter
  'help=The total number of accepted client connections.';

angie_connections_dropped $p8s_value
  path=/connections/dropped
  type=counter
  'help=The total number of dropped client connections.';

angie_connections_active $p8s_value
  path=/connections/active
  type=gauge
  'help=The current number of active client connections.';

angie_connections_idle $p8s_value
  path=/connections/idle
  type=gauge
  'help=The current number of idle client connections.';

'angie_slabs_pages_used{zone="$1"}' $p8s_value
  path=~~/slabs/([^/]+)/pages/used$
  type=gauge
  'help=The number of currently used memory pages in a slab zone.';

'angie_slabs_pages_free{zone="$1"}' $p8s_value
```



```

path=~~/slabs/([~/]+)/pages/free$
type=gauge
'help=The number of currently free memory pages in a slab zone.';

'angie_slabs_pages_slots_used{zone="$1",size="$2"}' $p8s_value
path=~~/slabs/([~/]+)/slots/([~/]+)/used$
type=gauge
'help=The number of currently used memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_free{zone="$1",size="$2"}' $p8s_value
path=~~/slabs/([~/]+)/slots/([~/]+)/free$
type=gauge
'help=The number of currently free memory slots of a specific size in a slab zone.';

'angie_slabs_pages_slots_reqs{zone="$1",size="$2"}' $p8s_value
path=~~/slabs/([~/]+)/slots/([~/]+)/reqs$
type=counter
'help=The total number of attempts to allocate a memory slot of a specific size in a slab_
↔zone.';

'angie_slabs_pages_slots_fails{zone="$1",size="$2"}' $p8s_value
path=~~/slabs/([~/]+)/slots/([~/]+)/fails$
type=counter
'help=The number of unsuccessful attempts to allocate a memory slot of a specific size in_
↔a slab zone.';

'angie_resolvers_queries{zone="$1",type="$2"}' $p8s_value
path=~~/resolvers/([~/]+)/queries/([~/]+)/$
type=counter
'help=The number of queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_sent{zone="$1",type="$2"}' $p8s_value
path=~~/resolvers/([~/]+)/sent/([~/]+)/$
type=counter
'help=The number of sent DNS queries of a specific type to resolve in a resolver zone.';

'angie_resolvers_responses{zone="$1",status="$2"}' $p8s_value
path=~~/resolvers/([~/]+)/responses/([~/]+)/$
type=counter
'help=The number of resolution results with a specific status in a resolver zone.';

'angie_http_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
path=~~/http/server_zones/([~/]+)/ssl/handshaked$
type=counter
'help=The total number of successful SSL handshakes in an HTTP server zone.';

```

```
'angie_http_server_zones_ssl_reuses{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/ssl/reuses$
  type=counter
  'help=The total number of session reuses during SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_timedout{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/ssl/timedout$
  type=counter
  'help=The total number of timed-out SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_ssl_failed{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/ssl/failed$
  type=counter
  'help=The total number of failed SSL handshakes in an HTTP server zone.';

'angie_http_server_zones_requests_total{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/requests/total$
  type=counter
  'help=The total number of client requests received in an HTTP server zone.';

'angie_http_server_zones_requests_processing{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/requests/processing$
  type=gauge
  'help=The number of client requests currently being processed in an HTTP server zone.';

'angie_http_server_zones_requests_discarded{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/requests/discarded$
  type=counter
  'help=The total number of client requests completed in an HTTP server zone without sending
↳ a response.';

'angie_http_server_zones_responses{zone="$1",code="$2"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/responses/([^/]+$)
  type=counter
  'help=The number of responses with a specific status in an HTTP server zone.';

'angie_http_server_zones_data_received{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/data/received$
  type=counter
  'help=The total number of bytes received from clients in an HTTP server zone.';

'angie_http_server_zones_data_sent{zone="$1"}' $p8s_value
  path=~^/http/server_zones/([^/]+)/data/sent$
  type=counter
```

```
'help=The total number of bytes sent to clients in an HTTP server zone.';

'angie_http_location_zones_requests_total{zone="$1"}' $p8s_value
path=~^/http/location_zones/([^/]+)/requests/total$
type=counter
'help=The total number of client requests in an HTTP location zone.';

'angie_http_location_zones_requests_discarded{zone="$1"}' $p8s_value
path=~^/http/location_zones/([^/]+)/requests/discarded$
type=counter
'help=The total number of client requests completed in an HTTP location zone without
↔sending a response.';

'angie_http_location_zones_responses{zone="$1",code="$2"}' $p8s_value
path=~^/http/location_zones/([^/]+)/responses/([^/]+$)
type=counter
'help=The number of responses with a specific status in an HTTP location zone.';

'angie_http_location_zones_data_received{zone="$1"}' $p8s_value
path=~^/http/location_zones/([^/]+)/data/received$
type=counter
'help=The total number of bytes received from clients in an HTTP location zone.';

'angie_http_location_zones_data_sent{zone="$1"}' $p8s_value
path=~^/http/location_zones/([^/]+)/data/sent$
type=counter
'help=The total number of bytes sent to clients in an HTTP location zone.';

'angie_http_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/state$
type=gauge
'help=The current state of an upstream peer in "HTTP": 1 - up, 2 - down, 3 - unavailable,
↔4 - recovering, 5 - unhealthy, 6 - checking, or 7 - draining.';

'angie_http_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/current$
type=gauge
'help=The number of requests currently being processed by an upstream peer in "HTTP".';

'angie_http_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
path=~^/http/upstreams/([^/]+)/peers/([^/]+)/selected/total$
type=counter
'help=The total number of attempts to use an upstream peer in "HTTP".';
```

```
'angie_http_upstreams_peers_responses{upstream="$1",peer="$2",code="$3"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/responses/([^/]+)$
    type=counter
    'help=The number of responses with a specific status received from an upstream peer in
↪"HTTP".';

'angie_http_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/sent$
    type=counter
    'help=The total number of bytes sent to an upstream peer in "HTTP".';

'angie_http_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/data/received$
    type=counter
    'help=The total number of bytes received from an upstream peer in "HTTP".';

'angie_http_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/fails$
    type=counter
    'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↪"HTTP".';

'angie_http_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/unavailable$
    type=counter
    'help=The number of times when an upstream peer in "HTTP" became "unavailable" due to
↪reaching the max_fails limit.';

'angie_http_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/downtime$
    type=counter
    'help=The total time (in milliseconds) that an upstream peer in "HTTP" was "unavailable".';

'angie_http_upstreams_peers_health_header_time{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/header_time$
    type=gauge
    'help=Average time (in milliseconds) to receive the response headers from an upstream peer
↪in "HTTP".';

'angie_http_upstreams_peers_health_response_time{upstream="$1",peer="$2"}' $p8s_value
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/response_time$
    type=gauge
    'help=Average time (in milliseconds) to receive the complete response from an upstream
↪peer in "HTTP".';
```

```
'angie_http_upstreams_peers_health_probes_count{upstream="$1",peer="$2"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/probes/count$
  type=counter
  'help=The total number of probes for this peer.';

'angie_http_upstreams_peers_health_probes_fails{upstream="$1",peer="$2"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/peers/([^/]+)/health/probes/fails$
  type=counter
  'help=The total number of failed probes for this peer.';

'angie_http_upstreams_keepalive{upstream="$1"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/keepalive$
  type=gauge
  'help=The number of currently cached keepalive connections for an HTTP upstream.';

'angie_http_upstreams_queue_queued{upstream="$1"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/queue/queued$
  type=counter
  'help=The total number of queued requests for an HTTP upstream.';

'angie_http_upstreams_queue_waiting{upstream="$1"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/queue/waiting$
  type=gauge
  'help=The number of requests currently waiting in an HTTP upstream queue.';

'angie_http_upstreams_queue_dropped{upstream="$1"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/queue/dropped$
  type=counter
  'help=The total number of requests dropped from an HTTP upstream queue because the client_
↳had prematurely closed the connection.';

'angie_http_upstreams_queue_timedout{upstream="$1"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/queue/timedout$
  type=counter
  'help=The total number of requests timed out from an HTTP upstream queue.';

'angie_http_upstreams_queue_overflows{upstream="$1"}' $p8s_value
  path=~^/http/upstreams/([^/]+)/queue/overflows$
  type=counter
  'help=The total number of requests rejected by an HTTP upstream queue because the size_
↳limit had been reached.';

'angie_http_caches_responses{zone="$1",status="$2"}' $p8s_value
  path=~^/http/caches/([^/]+)/([^/]+)/responses$
```

```
type=counter
'help=The total number of responses processed in an HTTP cache zone with a specific cache_
↔status.';

'angie_http_caches_bytes{zone="$1",status="$2"}' $p8s_value
path=~^/http/caches/([^/]+)/([^/]+)/bytes$
type=counter
'help=The total number of bytes processed in an HTTP cache zone with a specific cache_
↔status.';

'angie_http_caches_responses_written{zone="$1",status="$2"}' $p8s_value
path=~^/http/caches/([^/]+)/([^/]+)/responses_written$
type=counter
'help=The total number of responses written to an HTTP cache zone with a specific cache_
↔status.';

'angie_http_caches_bytes_written{zone="$1",status="$2"}' $p8s_value
path=~^/http/caches/([^/]+)/([^/]+)/bytes_written$
type=counter
'help=The total number of bytes written to an HTTP cache zone with a specific cache status.
↔';

'angie_http_caches_size{zone="$1"}' $p8s_value
path=~^/http/caches/([^/]+)/size$
type=gauge
'help=The current size (in bytes) of cached responses in an HTTP cache zone.';

'angie_http_caches_shards_size{zone="$1",path="$2"}' $p8s_value
path=~^/http/caches/([^/]+)/shards/([^/]+)/size$
type=gauge
'help=The current size (in bytes) of cached responses in a shard path of an HTTP cache_
↔zone.';

'angie_http_limit_conns{zone="$1",status="$2"}' $p8s_value
path=~^/http/limit_conns/([^/]+)/([^/]+)/$
type=counter
'help=The number of requests processed by an HTTP limit_conn zone with a specific result.';

'angie_http_limit_reqs{zone="$1",status="$2"}' $p8s_value
path=~^/http/limit_reqs/([^/]+)/([^/]+)/$
type=counter
'help=The number of requests processed by an HTTP limit_reqs zone with a specific result.';

'angie_stream_server_zones_ssl_handshaked{zone="$1"}' $p8s_value
```

```
path=~~/stream/server_zones/([~/]+)/ssl/handshaked$
type=counter
'help=The total number of successful SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_reuses{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/ssl/reuses$
type=counter
'help=The total number of session reuses during SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_timedout{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/ssl/timedout$
type=counter
'help=The total number of timed-out SSL handshakes in a stream server zone.';

'angie_stream_server_zones_ssl_failed{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/ssl/failed$
type=counter
'help=The total number of failed SSL handshakes in a stream server zone.';

'angie_stream_server_zones_connections_total{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/total$
type=counter
'help=The total number of client connections received in a stream server zone.';

'angie_stream_server_zones_connections_processing{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/processing$
type=gauge
'help=The number of client connections currently being processed in a stream server zone.';

'angie_stream_server_zones_connections_discarded{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/discarded$
type=counter
'help=The total number of client connections completed in a stream server zone without
↔establishing a session.';

'angie_stream_server_zones_connections_passed{zone="$1"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/connections/passed$
type=counter
'help=The total number of client connections in a stream server zone passed for handling
↔to a different listening socket.';

'angie_stream_server_zones_sessions{zone="$1",status="$2"}' $p8s_value
path=~~/stream/server_zones/([~/]+)/sessions/([~/]+)/$
type=counter
'help=The number of sessions finished with a specific status in a stream server zone.';
```

```
'angie_stream_server_zones_data_received{zone="$1"}' $p8s_value
  path=~~/stream/server_zones/([~/]+)/data/received$
  type=counter
  'help=The total number of bytes received from clients in a stream server zone.';

'angie_stream_server_zones_data_sent{zone="$1"}' $p8s_value
  path=~~/stream/server_zones/([~/]+)/data/sent$
  type=counter
  'help=The total number of bytes sent to clients in a stream server zone.';

'angie_stream_upstreams_peers_state{upstream="$1",peer="$2"}' $p8st_all_ups_state
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/state$
  type=gauge
  'help=The current state of an upstream peer in "stream": 1 - up, 2 - down, 3 - unavailable,
↔ 4 - recovering, 5 - unhealthy, 6 - checking, or 7 - draining.';

'angie_stream_upstreams_peers_selected_current{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/selected/current$
  type=gauge
  'help=The number of sessions currently being processed by an upstream peer in "stream".';

'angie_stream_upstreams_peers_selected_total{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/selected/total$
  type=counter
  'help=The total number of attempts to use an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_sent{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/data/sent$
  type=counter
  'help=The total number of bytes sent to an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_received{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/data/received$
  type=counter
  'help=The total number of bytes received from an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_pkt_sent{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/data/pkt_sent$
  type=counter
  'help=The total number of packets sent to an upstream peer in "stream".';

'angie_stream_upstreams_peers_data_pkt_received{upstream="$1",peer="$2"}' $p8s_value
  path=~~/stream/upstreams/([~/]+)/peers/([~/]+)/data/pkt_received$
  type=counter
```



```
'help=The total number of packets received from an upstream peer in "stream"';

'angie_stream_upstreams_peers_health_fails{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/fails$
type=counter
'help=The total number of unsuccessful attempts to communicate with an upstream peer in
↔"stream"';

'angie_stream_upstreams_peers_health_unavailable{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/unavailable$
type=counter
'help=The number of times when an upstream peer in "stream" became "unavailable" due to
↔reaching the max_fails limit.';

'angie_stream_upstreams_peers_health_downtime{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/downtime$
type=counter
'help=The total time (in milliseconds) that an upstream peer in "stream" was "unavailable".
↔';

'angie_stream_upstreams_peers_health_connect_time{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/connect_time$
type=gauge
'help=Average time (in milliseconds) to connect to an upstream peer in "stream"';

'angie_stream_upstreams_peers_health_first_byte_time{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/first_byte_time$
type=gauge
'help=Average time (in milliseconds) to receive the first byte from an upstream peer in
↔"stream"';

'angie_stream_upstreams_peers_health_last_byte_time{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/last_byte_time$
type=gauge
'help=Average time (in milliseconds) of the whole communication session with an upstream
↔peer in "stream"';

'angie_stream_upstreams_peers_health_probes_count{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/probes/count$
type=counter
'help=The total number of probes for this peer.';

'angie_stream_upstreams_peers_health_probes_fails{upstream="$1",peer="$2"}' $p8s_value
path=~^/stream/upstreams/([~/]+)/peers/([~/]+)/health/probes/fails$
type=counter
'help=The total number of failed probes for this peer.';
```

```
}  
  
map $p8s_value $p8st_all_ups_state {  
    volatile;  
    "up"          1;  
    "down"        2;  
    "unavailable" 3;  
    "recovering"  4;  
    "unhealthy"   5;  
    "checking"    6;  
    "draining"    7;  
    default       0;  
}
```

使用方法:

```
http {  
  
    include prometheus_all.conf;  
  
    #...  
  
    server {  
  
        listen 80;  
  
        location =/p8s {  
            prometheus all;  
        }  
  
        # ...  
  
    }  
}
```

```
$ curl localhost/p8s  
  
# Angie Prometheus template "all"  
...
```

## 指令

### prometheus

Syntax	<code>prometheus <i>template</i>;</code>
默认	—
Context	location

为 `location` 上下文设置处理程序模板; 该模板应由 `prometheus_template` 指令定义。在请求时, 此 `location` 评估并返回 Prometheus 格式的模板指标。

```
location =/p8s {  
    prometheus custom;  
}
```

```
$ curl localhost/p8s  
  
# Angie Prometheus template "custom"  
...
```

### prometheus\_template

Syntax	<code>prometheus_template <i>template</i> { ... }</code>
默认	—
Context	http

定义 Angie 收集和导出的指标的命名模板; 该模板应与 `prometheus` 指令一起使用。

#### 备注

Angie 还有一个预定义的 `all` 模板, 包含若干常见指标。

该模板可以包含任意数量的指标定义, 每个定义的结构如下: `<metric> <variable> [path=<match>] [type=<type>] [help=<help>];`

<b>metric</b>	设置在 Prometheus 格式响应中添加的指标名称。可以包含可选的标签定义 (...), 例如: <pre>http_requests_total{method="\$1",code="\$2"}</pre> 标签值可以使用 Angie 变量; 如果 <i>match</i> 是正则表达式, 也可以使用其捕获组与标签一起使用。在评估指标的 <i>variable</i> 本身时, 这些变量和组会被评估。
<b>variable</b>	设置要评估并添加到响应中的指标值的变量名称。如果没有这样的变量或结果值为空 (""), 则不会添加任何指标。

指标的值从 *variable* 中评估; 成功评估后, 指标将被添加到响应中, 例如:

```
'angie_time{version="$angie_version"}' $msec;
```

```
$ curl localhost/p8s
```

```
angie_time{version="1.8.0"} 1695119820.562
```

<b>path=match</b>	与 Angie API 的 <i>/status</i> 部分中的所有端到端指标路径匹配, 允许一次性将多个指标实例添加到响应中。
-------------------	---

在匹配过程中, 路径包括起始斜杠但不包括结束斜杠, 例如: */angie/generation*; 匹配是大小写不敏感的。有两种匹配模式:

**path=exact\_match** 逐字符匹配字符串。

**path=~regex\_match** 基于 PCRE 的匹配; 可以定义捕获组以与 *metric* 名称标签一起使用。

如果 *match* 匹配路径, 则此路径上的 Angie 指标值将存储在 *\$p8s\_value* 变量中, 可在设置 *path=* 时用作 *variable*。

使用正则表达式匹配时, 可能会发生多次匹配; 在这种情况下, 将为每个匹配路径添加一个指标到响应中。使用捕获组时, 这启用了一系列共享一个名称但具有不同标签的指标, 例如:

```
'angie_slabs_slots_free{zone="$1",size="$2"}' $p8s_value  
path=~~/slabs/([~/]+)/slots/([~/]+)/free$;
```

这为配置中包含的每个区域和大小组合添加一个指标:

```
angie_slabs_slots_free{zone="one",size="8"} 502  
angie_slabs_slots_free{zone="one",size="16"} 249  
angie_slabs_slots_free{zone="one",size="32"} 122  
angie_slabs_slots_free{zone="one",size="128"} 22  
angie_slabs_slots_free{zone="one",size="512"} 4  
angie_slabs_slots_free{zone="two",size="8"} 311  
...
```

如果没有匹配 (无论模式如何), 则不会添加任何指标。

#### 备注

`path=` 选项仅在 Angie 使用 *API* 模块构建时可用。

<code>type=type,</code> <code>help=help</code>	分别使用 Prometheus 格式 设置指标的类型和帮助字符串; 二者与指标一起添加到响应中, 无需任何转换或验证。
---	---

### 内置变量

`http_prometheus` 模块有一个内置变量, 当 Angie API 的 `/status` 部分中的路径与由 `prometheus_template` 指令定义的指标的 `match` 选项匹配时接收其值。

#### `$p8s_value`

如果由 `prometheus_template` 定义的指标的 `match` 匹配路径, 则此路径上的 Angie 指标值将存储在 `$p8s_value` 变量中。它旨在用作在基于 `path=` 的指标定义中的 `variable`。

存储在 `$p8s_value` 中的 Angie 指标值可能偏离 Prometheus 格式的要求。在这种情况下, `map` 指令可以帮助将字符串转换为数字:

```
map $p8s_value $ups_state_n {
    up           0;
    unavailable  1;
    down        2;
    default     3;
}

prometheus_template main {
    'angie_http_upstreams_state{upstream="$1",peer="$2}"' $ups_state_n
    path=~^/http/upstreams/([^/]+)/peers/([^/]+)/state$;
}
```

如果 Angie 指标是布尔类型, 即 `true` 或 `false`, 变量设置为 "1" 或 "0", 分别; 如果指标是 `null`, 变量设置为 "(null)"。对于日期值, 使用整数纪元格式。

## 代理

允许将请求传递到另一台（代理）服务器。

### 配置示例

```
location / {
    proxy_pass      http://localhost:8000;
    proxy_set_header Host      $host;
    proxy_set_header X-Real-IP $remote_addr;
}
```

## 指令

### proxy\_bind

语法	<code>proxy_bind address [transparent]   off;</code>
默认值	—
上下文	http, server, location

使发往代理服务器的出站连接从指定的本地 IP 地址和可选端口发起。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 `proxy_bind` 指令的效果，这允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许发往代理服务器的出站连接从非本地 IP 地址发起，例如，从客户端的真实 IP 地址：

```
proxy_bind $remote_addr transparent;
```

为了使此参数生效,通常要以[超级用户](#) 权限运行 Angie 工作进程。在 Linux 上,如果指定了 `transparent` 参数, 则不需要, 因为工作进程会从主进程继承 `CAP_NET_RAW` 能力。

#### ❗ 重要

必须配置内核路由表以拦截来自代理服务器的网络流量。

### proxy\_buffer\_size

语法	<code>proxy_buffer_size size;</code>
默认值	<code>proxy_buffer_size 4k 8k;</code>
上下文	http, server, location

设置用于读取从代理服务器接收到的响应的第一部分的缓冲区大小。这部分通常包含一个小的响应头。默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 取决于平台。不过, 它可以被设置得更小。

### proxy\_buffering

语法	<code>proxy_buffering on   off;</code>
默认值	<code>proxy_buffering on;</code>
上下文	http, server, location

启用或禁用对代理服务器响应的缓冲。

on	Angie 尽快从代理服务器接收响应, 并将其保存到由 <code>proxy_buffer_size</code> 和 <code>proxy_buffers</code> 指令设置的缓冲区中。如果整个响应不适合内存的一部分, 可以将其保存到磁盘上的临时文件。写入临时文件由 <code>proxy_max_temp_file_size</code> 和 <code>proxy_temp_file_write_size</code> 指令控制。
off	响应以同步方式传递给客户端, 立即在收到时传递。Angie 不会尝试从代理服务器读取整个响应。Angie 从服务器一次可以接收的数据的最大大小由 <code>proxy_buffer_size</code> 指令设置。

可以通过在“X-Accel-Buffering”响应头字段中传递“yes”或“no”来启用或禁用缓冲。此功能可以使用 `proxy_ignore_headers` 指令禁用。

### proxy\_buffers

语法	<code>proxy_buffers number size;</code>
默认值	<code>proxy_buffers 8 4k   8k;</code>
上下文	http, server, location

设置用于读取代理服务器响应的缓冲区的数量和大小, 针对单个连接。

默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 取决于平台。

### proxy\_busy\_buffers\_size

语法	<code>proxy_busy_buffers_size size;</code>
默认值	<code>proxy_busy_buffers_size 8k   16k;</code>
上下文	http, server, location

当从代理服务器的响应的缓冲 被启用时, 限制可以用于发送给客户端的响应但尚未完全读取的缓冲区总大小。同时, 剩余的缓冲区可用于读取响应, 并在需要时将部分响应缓冲到临时文件。

默认情况下, 大小由`proxy_buffer_size` 和`proxy_buffers` 指令设置的两个缓冲区的大小限制。

### proxy\_cache

语法	<code>proxy_cache zone   off [path=path];</code>
默认值	<code>proxy_cache off;</code>
上下文	http, server, location

定义用于缓存的共享内存区域。一个区域可以在配置中多次使用。参数的值允许变量。

off	禁用从先前配置级别继承的缓存。
-----	-----------------

Added in version 1.2.0: PRO

在 Angie PRO 中, 您可以指定多个共享相同 `keys_zone` 值的`proxy_cache_path` 指令以实现缓存分片。如果这样做, 请设置引用此 `keys_zone` 的`proxy_cache` 指令的 `path` 参数:

<code>path=path</code>	该值在后台的响应被缓存时确定, 这意味着涉及变量, 包括那些存储来自响应的信息的变量。 如果响应从缓存中获得, <code>path</code> 不会重新评估; 因此, 从缓存中获取的响应将保留其原始 <code>path</code> , 直到从缓存中删除。
------------------------	--

这允许通过应用`map` 指令或脚本来选择缓存路径, 以响应来自后台的响应。一个 Content-Type 示例:

```
proxy_cache_path /cache/one keys_zone=zone:10m;
proxy_cache_path /cache/two keys_zone=zone;

map $upstream_http_content_type $cache {
    ~^text/ one;
    default two;
}

server {
```



```
...
location / {
    proxy_pass http://backend;
    proxy_cache zone path=/cache/$cache;
}
}
```

这添加了两个缓存路径和一个变量映射以选择它们之间。如果 Content-Type 以 text/ 开头, 则使用第一个路径; 否则, 使用第二个。

### proxy\_cache\_background\_update

语法	proxy_cache_background_update on   off;
默认值	proxy_cache_background_update off;
上下文	http, server, location

允许启动一个后台子请求以更新已过期的缓存项, 同时将陈旧的缓存响应返回给客户端。

#### ⚠ 注意

请注意, 当缓存响应正在更新时, 必须允许使用陈旧的缓存响应。

### proxy\_cache\_bypass

语法	proxy_cache_bypass ...;
默认值	—
上下文	http, server, location

定义在何种条件下不会从缓存中获取响应。如果字符串参数的至少一个值不为空且不等于“0”, 则不会从缓存中获取响应:

```
proxy_cache_bypass $cookie_nocache $arg_nocache$arg_comment;
proxy_cache_bypass $http_pragma $http_authorization;
```

可以与 `proxy_no_cache` 指令一起使用。

### proxy\_cache\_convert\_head

语法	proxy_cache_convert_head on   off;
默认值	proxy_cache_convert_head on;
上下文	http, server, location

启用或禁用将”HEAD”方法转换为”GET”以进行缓存。当转换被禁用时,应该配置缓存键以包含`$request_method`。

### proxy\_cache\_key

语法	proxy_cache_key <i>string</i> ;
默认值	proxy_cache_key \$scheme\$proxy_host\$request_uri;
上下文	http, server, location

定义缓存的键,例如

```
proxy_cache_key "$host$request_uri $cookie_user";
```

默认情况下,指令的值接近于字符串

```
proxy_cache_key $scheme$proxy_host$uri$is_args$args;
```

### proxy\_cache\_lock

语法	proxy_cache_lock on   off;
默认值	proxy_cache_lock off;
上下文	http, server, location

启用时,只允许一个请求通过请求代理服务器填充由`proxy_cache_key`指令识别的新缓存元素。其他相同缓存元素的请求将等待响应出现在缓存中或此元素的缓存锁被释放,直到由`proxy_cache_lock_timeout`指令设置的时间。

### proxy\_cache\_lock\_age

语法	<code>proxy_cache_lock_age time;</code>
默认值	<code>proxy_cache_lock_age 5s;</code>
上下文	http, server, location

如果最后一个传递到代理服务器以填充新缓存元素的请求未在指定时间内完成, 则可以再传递一个请求到代理服务器。

### proxy\_cache\_lock\_timeout

语法	<code>proxy_cache_lock_timeout time;</code>
默认值	<code>proxy_cache_lock_timeout 5s;</code>
上下文	http, server, location

设置`proxy_cache_lock`的超时时间。当时间到期时, 请求将被传递到代理服务器, 但响应不会被缓存。

### proxy\_cache\_max\_range\_offset

语法	<code>proxy_cache_max_range_offset number;</code>
默认值	—
上下文	http, server, location

设置字节范围请求的偏移量。如果范围超出偏移量, 范围请求将被传递到代理服务器, 并且响应不会被缓存。

### proxy\_cache\_methods

语法	<code>proxy_cache_methods GET   HEAD   POST ...;</code>
默认值	<code>proxy_cache_methods GET HEAD;</code>
上下文	http, server, location

如果客户端请求方法在此指令中列出, 则响应将被缓存。"GET" 和"HEAD" 方法总是被添加到列表中, 尽管建议显式指定它们。另请参见`proxy_no_cache` 指令。

### proxy\_cache\_min\_uses

语法	<code>proxy_cache_min_uses number;</code>
默认值	<code>proxy_cache_min_uses 1;</code>
上下文	http, server, location

设置请求次数, 在达到该次数后响应将被缓存。

### proxy\_cache\_path

语法	<code>proxy_cache_path path [levels=levels] [use_temp_path=on   off] keys_zone=name:size [inactive=time] [max_size=size] [min_free=size] [manager_files=number] [manager_sleep=time] [manager_threshold=time] [loader_files=number] [loader_sleep=time] [loader_threshold=time];</code>
默认值	—
上下文	http

设置缓存的 *path* 和其他参数。缓存数据存储在文件中。缓存中的文件名是应用 MD5 函数到 *cache key* 的结果。

<b>levels</b>	定义缓存的层次结构级别: 从 1 到 3, 每个级别接受值 1 或 2。
---------------	--------------------------------------

例如, 在以下配置中:

```
proxy_cache_path /data/angie/cache levels=1:2 keys_zone=one:10m;
```

缓存中的文件名将如下所示:

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存的响应首先写入临时文件, 然后文件被重命名。临时文件和缓存可以放在不同的文件系统上。但是请注意, 在这种情况下, 文件跨两个文件系统复制, 而不是廉价的重命名操作。因此建议对任何给定位置, 缓存和保存临时文件的目录都放在同一文件系统上。

<code>use_temp_path=on</code>	设置临时文件的目录
<code>off</code>	
<code>on</code>	如果此参数被省略或设置为值 <code>on</code> , 则使用 <code>proxy_temp_path</code> 指令为给定 <code>location</code> 设置的目录。
<code>off</code>	临时文件将直接放在缓存目录中。
<code>keys_zone</code>	配置共享内存区的名称和大小, 以存储所有活动键和数据信息。 一兆字节的区域可以存储大约 8,000 个键。
<code>inactive</code>	在此参数指定的时间内未被访问的缓存数据将从缓存中移除, 无论其新鲜度如何。 默认情况下, 设置为 10 分钟。

**i 备注**

Added in version 1.2.0: PRO

在 Angie PRO 中, 允许多个 `proxy_cache_path` 指令共享相同的 `keys_zone` 值。只有第一个这样的指令可以设置共享内存区大小。对这些指令的选择由相关 `proxy_cache` 指令的 `path` 参数决定。

一个特殊的 **cache manager** 进程监控文件系统上的最大缓存大小和最小可用空间, 当大小超出或没有足够的可用空间时, 它会删除最近最少使用的数据。这些数据以迭代的方式删除。

<code>max_size</code>	最大缓存大小
<code>min_free</code>	文件系统上缓存的最小可用空间
<code>manager_files</code>	限制一次迭代中要删除的项目数量 默认情况下, 100
<code>manager_threshol</code>	限制一次迭代的持续时间 默认情况下, 200 毫秒
<code>manager_sleep</code>	配置交互之间的暂停 默认情况下, 50 毫秒

Angie 启动一分钟后, 特殊的 **cache loader** 进程被激活。它加载文件系统中存储的先前缓存数据的信息到缓存区。加载同样以迭代的方式进行。

<code>loader_files</code>	限制一次迭代中要加载的项目数量 默认情况下, 100
<code>loader_threshold</code>	限制一次迭代的持续时间 默认情况下, 200 毫秒
<code>loader_sleep</code>	配置交互之间的暂停 默认情况下, 50 毫秒

### proxy\_cache\_revalidate

语法	<code>proxy_cache_revalidate on   off;</code>
默认	<code>proxy_cache_revalidate off;</code>
上下文	http, server, location

启用使用带有”If-Modified-Since” 和”If-None-Match” 头字段的条件请求重新验证过期的缓存项。

### proxy\_cache\_use\_stale

语法	<code>proxy_cache_use_stale error   timeout   invalid_header   updating   http_500   http_502   http_503   http_504   http_403   http_404   http_429   off ...;</code>
默认	<code>proxy_cache_use_stale off;</code>
上下文	http, server, location

决定在与代理服务器通信的哪些情况下可以使用陈旧的缓存响应。指令的参数与`proxy_next_upstream` 指令的参数匹配。

<code>error</code>	如果无法选择处理请求的代理服务器, 则允许使用陈旧的缓存响应。
<code>updating</code>	额外参数, 允许使用正在更新的陈旧缓存响应。这允许在更新缓存数据时最小化对代理服务器的访问次数。

直接在响应头中也可以为响应过期后指定的秒数启用使用陈旧缓存响应:

- ”Cache-Control” 头字段的 `stale-while-revalidate` 扩展允许使用正在更新的陈旧缓存响应。
- ”Cache-Control” 头字段的 `stale-if-error` 扩展允许在出现错误时使用陈旧的缓存响应。

#### **i** 备注

这比使用指令参数的优先级更低。

为最小化在填充新缓存元素时对代理服务器的访问次数, 可以使用`proxy_cache_lock` 指令。

## proxy\_cache\_valid

语法	<code>proxy_cache_valid [code ...] time;</code>
默认	—
上下文	http, server, location

设置不同响应代码的缓存时间。例如, 以下指令

```
proxy_cache_valid 200 302 10m;  
proxy_cache_valid 404 1m;
```

为代码 200 和 302 的响应设置 10 分钟的缓存时间, 为代码 404 的响应设置 1 分钟的缓存时间。

如果只指定缓存时间

```
proxy_cache_valid 5m;
```

则仅缓存 200、301 和 302 响应。

此外, 可以指定 `any` 参数以缓存任何响应:

```
proxy_cache_valid 200 302 10m;  
proxy_cache_valid 301 1h;  
proxy_cache_valid any 1m;
```

### 备注

缓存的参数也可以直接在响应头中设置。这比使用指令设置缓存时间的优先级更高。

- "X-Accel-Expires" 头字段以秒为单位设置响应的缓存时间。值为零禁用响应的缓存。如果值以 @ 前缀开头, 则设置自 Epoch 以来的绝对时间, 以秒为单位, 直到响应可能被缓存。
- 如果头字段不包含 "X-Accel-Expires" 字段, 缓存参数可能会在 "Expires" 或 "Cache-Control" 头字段中设置。
- 如果头字段包含 "Set-Cookie" 字段, 则该响应将不会被缓存。
- 如果头字段包含值为 "\*" 的 "Vary" 字段, 则该响应将不会被缓存。如果头字段包含其他值的 "Vary" 字段, 则此类响应将被缓存, 并考虑相应的请求头字段。

可以使用 `proxy_ignore_headers` 指令禁用对这些响应头字段中的一个或多个的处理。

### proxy\_connect\_timeout

语法	<code>proxy_connect_timeout time;</code>
默认	<code>proxy_connect_timeout 60s;</code>
上下文	http, server, location

定义与代理服务器建立连接的超时时间。应注意, 此超时时间通常不能超过 75 秒。

### proxy\_connection\_drop

语法	<code>proxy_connection_drop time   on   off;</code>
默认	<code>proxy_connection_drop off;</code>
上下文	http, server, location

启用在代理服务器从组中移除或被 *eresolve* 进程或 *API command* DELETE 标记为永久不可用后, 终止与其的所有连接。

当为客户端或代理服务器处理下一个读取或写入事件时, 连接被终止。

设置 *time* 启用连接终止 timeout; 设置 on 时, 连接立即断开。

### proxy\_cookie\_domain

- 语法
  - `proxy_cookie_domain off;`
  - `proxy_cookie_domain domain replacement;`
- 默认
  - `proxy_cookie_domain off;`
- 上下文
  - http, server, location

设定应在代理服务器响应的“Set-Cookie”头字段的域属性中更改的文本。假设代理服务器返回的“Set-Cookie”头字段具有属性“domain=localhost”。指令

```
proxy_cookie_domain localhost example.org;
```

将这个属性重写为“domain=example.org”。

忽略 *domain* 和 *replacement* 字符串以及域属性开头的点号。匹配不区分大小写。

*domain* 和 *replacement* 字符串可以包含变量:



```
proxy_cookie_domain www.$host $host;
```

该指令也可以使用正则表达式指定。在这种情况下, *domain* 应该以 “~” 符号开始。正则表达式可以包含命名和位置捕获, 而 *replacement* 可以引用它们:

```
proxy_cookie_domain ~\.(?P<sl_domain>[-0-9a-z]+\.[a-z]+)$ $sl_domain;
```

可以在同一级别指定多个 *proxy\_cookie\_domain* 指令:

```
proxy_cookie_domain localhost example.org;  
proxy_cookie_domain ~\.[a-z]+\.[a-z]+)$ $1;
```

如果可以将多个指令应用于 *cookie*, 将选择第一个匹配的指令。

*off* 参数取消从先前配置级别继承的 *proxy\_cookie\_domain* 指令的效果。

### proxy\_cookie\_flags

语法	<code>proxy_cookie_flags off   cookie [flag ...];</code>
默认	<code>proxy_cookie_flags off;</code>
上下文	http, server, location

为 *cookie* 设置一个或多个标志。 *cookie* 可以包含文本、变量及其组合。标志可以包含文本、变量及其组合。

*secure*、*httponly*、*samesite=strict*、*samesite=lax*、*samesite=none* 参数添加相应的标志。

*nosecure*、*nohttponly*、*nosamesite* 参数移除相应的标志。

*cookie* 也可以使用正则表达式指定。在这种情况下, *cookie* 应该以 “~” 符号开始。

可以在同一配置级别指定多个 *proxy\_cookie\_flags* 指令:

```
proxy_cookie_flags one httponly;  
proxy_cookie_flags ~ nosecure samesite=strict;
```

如果可以将多个指令应用于 *cookie*, 将选择第一个匹配的指令。在此示例中, *httponly* 标志被添加到 *cookie one*, 而对所有其他 *cookie*, *samesite=strict* 标志被添加, *secure* 标志被删除。

*off* 参数取消从先前配置级别继承的 *proxy\_cookie\_flags* 指令的效果。

## proxy\_cookie\_path

语法	<code>proxy_cookie_path off;</code> <code>proxy_cookie_path path replacement;</code>
默认	<code>proxy_cookie_path off;</code>
上下文	http, server, location

设定应在代理服务器响应的 `Set-Cookie` 头字段的路径属性中更改的文本。假设代理服务器返回的“Set-Cookie”头字段具有属性“`path=/two/some/uri/`”。指令

```
proxy_cookie_path /two/ /;
```

将这个属性重写为“`path=/some/uri/`”。

`path` 和 `replacement` 字符串可以包含变量:

```
proxy_cookie_path $uri /some$uri;
```

该指令也可以使用正则表达式指定。在这种情况下, `path` 应该以“`~`”符号开始以进行区分大小写的匹配, 或以“`~*`”符号开始以进行不区分大小写的匹配。正则表达式可以包含命名和位置捕获, 而 `replacement` 可以引用它们:

```
proxy_cookie_path ~*/user/([~/]+) /u/$1;
```

可以在同一级别指定多个 `proxy_cookie_path` 指令:

```
proxy_cookie_path /one/ /;  
proxy_cookie_path / /two/;
```

如果可以将多个指令应用于 cookie, 将选择第一个匹配的指令。

`off` 参数取消从先前配置级别继承的 `proxy_cookie_path` 指令的效果。

## proxy\_force\_ranges

语法	<code>proxy_force_ranges off;</code>
默认	<code>proxy_force_ranges off;</code>
上下文	http, server, location

启用代理服务器响应中缓存和未缓存响应的字节范围支持, 而不管这些响应中的“Accept-Ranges”字段。

### proxy\_headers\_hash\_bucket\_size

语法	<code>proxy_headers_hash_bucket_size size;</code>
默认	<code>proxy_headers_hash_bucket_size 64;</code>
上下文	http, server, location

设置 `proxy_hide_header` 和 `proxy_set_header` 指令所使用的哈希表的桶大小。有关设置哈希表的详细信息, 请参阅 [单独说明](#)。

### proxy\_headers\_hash\_max\_size

语法	<code>proxy_headers_hash_max_size size;</code>
默认	<code>proxy_headers_hash_max_size 512;</code>
上下文	http, server, location

设置 `proxy_hide_header` 和 `proxy_set_header` 指令所使用的哈希表的最大大小。有关设置哈希表的详细信息, 请参阅 [单独说明](#)。

### proxy\_hide\_header

语法	<code>proxy_hide_header field;</code>
默认	—
上下文	http, server, location

默认情况下, Angie 不会将代理服务器响应中的“Date”、“Server”、“X-Pad”和“X-Accel-...”头字段传递给客户端。`proxy_hide_header` 指令设置不传递的其他字段。如果需要允许传递字段, 可以使用 `proxy_pass_header` 指令。

### proxy\_http\_version

语法	<code>proxy_http_version 1.0   1.1   3;</code>
默认	<code>proxy_http_version 1.0;</code>
上下文	http, server, location, if in location, limit_except

为代理设置 HTTP 协议版本。默认情况下, 使用版本 1.0。建议使用 1.1 或更高版本与 [保持活动连接](#) 一起使用。

### proxy\_http3\_hq

语法	<code>proxy_http3_hq on   off;</code>
默认	<code>proxy_http3_hq off;</code>
上下文	<code>http, server</code>

切换特定的 hq-interop 协商模式, 该模式用于 *QUIC* 互操作性测试 Angie 依赖的。

### proxy\_http3\_max\_concurrent\_streams

语法	<code>proxy_http3_max_concurrent_streams number;</code>
默认	<code>proxy_http3_max_concurrent_streams 128;</code>
上下文	<code>http, server</code>

设置连接 中最大并发 HTTP/3 请求流的数量。需要启用保持活动连接。

### proxy\_http3\_stream\_buffer\_size

语法	<code>proxy_http3_stream_buffer_size size;</code>
默认	<code>proxy_http3_stream_buffer_size 64k;</code>
上下文	<code>http, server</code>

设置用于 *QUIC* 流的读写缓冲区的大小。

### proxy\_ignore\_client\_abort

语法	<code>proxy_ignore_client_abort on   off;</code>
默认	<code>proxy_ignore_client_abort off;</code>
上下文	<code>http, server, location</code>

确定当客户端在未等待响应的情况下关闭连接时, 是否应关闭与代理服务器的连接。

### proxy\_ignore\_headers

语法	<code>proxy_ignore_headers field ...;</code>
默认	—
上下文	http, server, location

禁用来自代理服务器的某些响应头字段的处理。可以忽略以下字段：“X-Accel-Redirect”、“X-Accel-Expires”、“X-Accel-Limit-Rate”、“X-Accel-Buffering”、“X-Accel-Charset”、“Expires”、“Cache-Control”、“Set-Cookie”和“Vary”。

如果没有被禁用，这些头字段的处理将产生以下效果：

- “X-Accel-Expires”、“Expires”、“Cache-Control”、“Set-Cookie”和“Vary”设置响应的缓存参数；
- “X-Accel-Redirect”执行到指定 URI 的内部重定向；
- “X-Accel-Limit-Rate”设置将响应传输给客户端的速率限制；
- “X-Accel-Buffering”启用或禁用响应的缓冲；
- “X-Accel-Charset”设置响应的所需字符集。

### proxy\_intercept\_errors

语法	<code>proxy_intercept_errors on   off;</code>
默认	<code>proxy_intercept_errors off;</code>
上下文	http, server, location

确定是否应将代码大于或等于 300 的代理响应传递给客户端，还是被拦截并重定向到 `Angie` 进行使用 `error_page` 指令的处理。

### proxy\_limit\_rate

语法	<code>proxy_limit_rate rate;</code>
默认值	<code>proxy_limit_rate 0;</code>
上下文	http, server, location

限制从代理服务器读取响应的速度。`rate` 以每秒字节数指定，并且可以包含变量。

0	禁用速率限制
---	--------

**i 备注**

限制是针对每个请求设置的, 因此如果 `Angie` 同时打开两个到代理服务器的连接, 总速率将是指定限制的两倍。此限制仅在启用从代理服务器缓冲响应的情况下有效。

**proxy\_max\_temp\_file\_size**

语法	<code>proxy_max_temp_file_size size;</code>
默认值	<code>proxy_max_temp_file_size 1024m;</code>
上下文	<code>http, server, location</code>

当启用从代理服务器缓冲响应, 并且整个响应无法容纳在由 `proxy_buffer_size` 和 `proxy_buffers` 指令设置的缓冲区中时, 响应的一部分可以保存到临时文件中。此指令设置临时文件的最大大小。每次写入临时文件的数据大小由 `proxy_temp_file_write_size` 指令设置。

0	禁用对临时文件的响应缓冲
---	--------------

**i 备注**

此限制不适用于将被缓存或存储在磁盘上的响应。

**proxy\_method**

语法	<code>proxy_method method;</code>
默认值	—
上下文	<code>http, server, location</code>

指定转发到代理服务器的请求中使用的 HTTP 方法, 而不是客户端请求中的方法。参数值可以包含变量。

**proxy\_next\_upstream**

语法	<code>proxy_next_upstream error   timeout   invalid_header   http_500   http_502   http_503   http_504   http_403   http_404   http_429   non_idempotent   off ...;</code>
默认值	<code>proxy_next_upstream error timeout;</code>
上下文	<code>http, server, location</code>

指定在何种情况下应将请求传递到上游池中的下一个服务器:

error	建立与服务器的连接、传递请求或读取响应头时发生错误;
timeout	建立与服务器的连接、传递请求或读取响应头时发生超时;
invalid_header	服务器返回了一个空或无效的响应;
http_500	服务器返回了代码为 500 的响应;
http_502	服务器返回了代码为 502 的响应;
http_503	服务器返回了代码为 503 的响应;
http_504	服务器返回了代码为 504 的响应;
http_403	服务器返回了代码为 403 的响应;
http_404	服务器返回了代码为 404 的响应;
http_429	服务器返回了代码为 429 的响应;
non_idempotent	通常, 具有 <b>非幂等</b> 方法 (POST, LOCK, PATCH) 的请求不会传递到下一个服务器, 如果请求已发送到上游服务器; 启用此选项显式允许重试此类请求;
off	禁止将请求传递到下一个服务器。

**i** 备注

需要注意的是, 只有在尚未向客户端发送任何内容的情况下, 才能将请求传递到下一个服务器。也就是说, 如果在传输响应的中途发生错误或超时, 修复是无法实现的。

该指令还定义了与服务器通信的**不成功尝试**的标准。

error	始终被视为不成功的尝试, 即使它们未在指令中指定
timeout	
invalid_header	
http_500	仅在指令中指定时被视为不成功的尝试
http_502	
http_503	
http_504	
http_429	
http_403	从不被视为不成功的尝试
http_404	

将请求传递到下一个服务器可以通过**尝试次数**和**时间**进行限制。

### proxy\_next\_upstream\_timeout

语法	<code>proxy_next_upstream_timeout time;</code>
默认值	<code>proxy_next_upstream_timeout 0;</code>
上下文	http, server, location

限制请求可以传递到下一个 服务器的时间。

0	关闭此限制
---	-------

### proxy\_next\_upstream\_tries

语法	<code>proxy_next_upstream_tries number;</code>
默认值	<code>proxy_next_upstream_tries 0;</code>
上下文	http, server, location

限制将请求传递到下一个 服务器的可能尝试次数。

0	关闭此限制
---	-------

### proxy\_no\_cache

语法	<code>proxy_no_cache string ...;</code>
默认值	—
上下文	http, server, location

定义在何种条件下响应不会被保存到缓存中。如果字符串参数的至少一个值不为空且不等于”0”，则响应不会被保存：

```
proxy_no_cache $cookie_nocache $arg_nocache$arg_comment;  
proxy_no_cache $http_pragma $http_authorization;
```

可以与`proxy_cache_bypass` 指令一起使用。



## proxy\_pass

语法	<code>proxy_pass uri;</code>
默认值	—
上下文	location, if in location

设置代理服务器的协议和地址以及一个可选的 URI, 以将位置映射到该 URI。作为协议, 可以指定 `http` 或 `https`。地址可以指定为域名或 IP 地址, 并且可以选择性地指定端口:

```
proxy_pass http://localhost:8000/uri/;
```

或者作为 UNIX 域套接字路径, 在 `unix` 之后并用冒号括起来指定:

```
proxy_pass http://unix:/tmp/backend.socket:/uri/;
```

如果一个域名解析为多个地址, 则所有这些地址将以轮询方式使用。此外, 地址可以指定为服务器组。如果使用组, 则不能与其一起指定端口; 相反, 应为组内的每个服务器单独指定端口。

参数值可以包含变量。在这种情况下, 如果地址指定为域名, 则在描述的服务器组中搜索该名称, 如果未找到, 则使用 `resolver` 确定。

请求 URI 被传递到服务器的方式如下:

- 如果 `proxy_pass` 指令是带 URI 指定的, 则当请求传递到服务器时, 与位置匹配的标准化的请求 URI 的部分将被指令中指定的 URI 替换:

```
location /name/ {  
    proxy_pass http://127.0.0.1/remote/;  
}
```

- 如果 `proxy_pass` 是不带 URI 指定的, 则请求 URI 以客户端发送的原始形式传递到服务器, 或在处理更改后的 URI 时传递完整的标准化请求 URI:

```
location /some/path/ {  
    proxy_pass http://127.0.0.1;  
}
```

在某些情况下, 无法确定要替换的请求 URI 部分:

- 当 `location` 使用正则表达式指定时, 以及在命名的 `locations` 内

在这些情况下, `proxy_pass` 应该指定为不带 URI。

- 当使用 `rewrite` 指令在代理的 `location` 内更改 URI, 并且将使用相同的配置处理请求时 (`break`):

```
location /name/ {  
    rewrite /name/([^/]+)/users?name=$1 break;  
    proxy_pass http://127.0.0.1;  
}
```

在这种情况下, 指令中指定的 URI 将被忽略, 完整的更改后的请求 URI 将传递到服务器。

- 当在 `proxy_pass` 中使用变量时:

```
location /name/ {  
    proxy_pass http://127.0.0.1$request_uri;  
}
```

在这种情况下, 如果在指令中指定了 URI, 则它将按原样传递到服务器, 替换原始请求 URI。

`WebSocket` 代理需要特殊配置。

### proxy\_pass\_header

语法	<code>proxy_pass_header field ...;</code>
默认值	—
上下文	http, server, location

允许从代理服务器传递否则禁用的 头字段到客户端。

### proxy\_pass\_request\_body

语法	<code>proxy_pass_request_body on   off;</code>
默认值	<code>proxy_pass_request_body on;</code>
上下文	http, server, location

指示是否将原始请求体传递到代理服务器。

```
location /x-accel-redirect-here/ {  
    proxy_method GET;  
    proxy_pass_request_body off;  
    proxy_set_header Content-Length "";  
  
    proxy_pass ...;  
}
```

另请参见 `proxy_set_header` 和 `proxy_pass_request_headers` 指令。

### proxy\_pass\_request\_headers

语法	proxy_pass_request_headers on   off;
默认值	proxy_pass_request_headers on;
上下文	http, server, location

指示是否将原始请求的头字段传递给代理服务器。

```
location /x-accel-redirect-here/ {
    proxy_method GET;
    proxy_pass_request_headers off;
    proxy_pass_request_body off;

    proxy_pass ...;
}
```

另请参见 `proxy_set_header` 和 `proxy_pass_request_body` 指令。

### proxy\_quic\_active\_connection\_id\_limit

语法	proxy_quic_active_connection_id_limit <i>number</i> ;
默认值	proxy_quic_active_connection_id_limit 2;
上下文	http, server

设置 *QUIC* `active_connection_id_limit` 传输参数值。这是每个服务器可以维护的最大活动 `连接 ID` 数量。

### proxy\_quic\_gso

语法	proxy_quic_gso on   off;
默认值	proxy_quic_gso off;
上下文	http, server

切换使用 (通用分段卸载) 在 *QUIC* 优化的批处理模式下发送数据。

### proxy\_quic\_host\_key

语法	<code>proxy_quic_host_key file;</code>
默认值	—
上下文	http, server

设置一个 *file*, 其中包含用于 *QUIC* 的密钥, 用于加密 无状态重置 和 地址验证 令牌。默认情况下, 每次重启时会生成一个随机密钥。使用旧密钥生成的令牌将不被接受。

### proxy\_read\_timeout

语法	<code>proxy_read_timeout time;</code>
默认值	<code>proxy_read_timeout 60s;</code>
上下文	http, server, location

定义从代理服务器读取响应的超时时间。超时仅在两个连续的读取操作之间设置, 而不是用于整个响应的传输。如果代理服务器在此时间内未传输任何内容, 则连接将关闭。

### proxy\_redirect

语法	<code>proxy_redirect default;</code> <code>proxy_redirect off;</code> <code>proxy_redirect redirect replacement;</code>
默认值	<code>proxy_redirect default;</code>
上下文	http, server, location

设置应更改代理服务器响应的”Location” 和”Refresh” 头字段中的文本。

假设代理服务器返回了头字段:

```
Location: http://localhost:8000/two/some/uri/
```

该指令

```
proxy_redirect http://localhost:8000/two/ http://frontend/one/;
```

将把此字符串重写为:

```
Location: http://frontend/one/some/uri/
```

在 *replacement* 字符串中可以省略服务器名称:

```
proxy_redirect http://localhost:8000/two/ /;
```

然后主服务器的名称和端口（如果不同于 80）将被插入。

`default` 参数指定的默认替换使用 `location` 和 `proxy_pass` 指令的参数。因此，以下两个配置是等效的：

```
location /one/ {
    proxy_pass      http://upstream:port/two/;
    proxy_redirect default;
```

```
location /one/ {
    proxy_pass      http://upstream:port/two/;
    proxy_redirect http://upstream:port/two/ /one/;
```

### ⚠️ 小心

如果 `proxy_pass` 是使用变量指定的，则不允许使用默认参数。

`replacement` 字符串可以包含变量：

```
proxy_redirect http://localhost:8000/ http://$host:$server_port/;
```

`redirect` 也可以包含变量：

```
proxy_redirect http://$proxy_host:8000/ /;
```

可以使用正则表达式指定该指令。在这种情况下，`redirect` 应以“~”符号开头用于区分大小写匹配，或以“~\*”符号开头用于不区分大小写匹配。正则表达式可以包含命名和位置捕获，`replacement` 可以引用它们：

```
proxy_redirect ~^(http://[^\:]+\d+(\./.)$ $1$2;
proxy_redirect ~*/user/([^\:]+)/(.+)$ http://$1.example.com/$2;
```

可以在同一层次上指定多个 `proxy_redirect` 指令：

```
proxy_redirect default;
proxy_redirect http://localhost:8000/ /;
proxy_redirect http://www.example.com/ /;
```

如果可以将多个指令应用于代理服务器响应的头字段，则会选择第一个匹配的指令。

`off` 参数取消继承自前一级配置的 `proxy_redirect` 指令的效果。

使用此指令，还可以将主机名称添加到代理服务器发出的相对重定向：

```
proxy_redirect / /;
```

### proxy\_request\_buffering

语法	<code>proxy_request_buffering on   off;</code>
默认值	<code>proxy_request_buffering on;</code>
上下文	http, server, location

启用或禁用客户端请求主体的缓冲。

on	整个请求主体在发送给代理服务器之前从客户端读取。
off	请求主体在接收到时立即发送到代理服务器。在这种情况下, 如果代理服务器已经开始发送请求主体, 则请求无法传递给下一个服务器。

当使用 HTTP/1.1 分块传输编码发送原始请求主体时, 无论指令值如何, 请求主体都将被缓冲, 除非 HTTP/1.1 已为代理启用 *QUIC*。

### proxy\_send\_lowat

语法	<code>proxy_send_lowat size;</code>
默认值	<code>proxy_send_lowat 0;</code>
上下文	http, server, location

如果该指令设置为非零值, 代理将尝试通过使用 *kqueue* 方法的 *NOTE\_LOWAT* 标志或 *SO\_SNDLOWAT* 套接字选项, 以指定的大小来最小化对代理服务器的传出连接的发送操作次数。

#### 备注

在 Linux、Solaris 和 Windows 上忽略此指令。

### proxy\_send\_timeout

语法	<code>proxy_send_timeout time;</code>
默认值	<code>proxy_send_timeout 60s;</code>
上下文	http, server, location

设置将请求传输到代理服务器的超时时间。超时仅在两个连续的写操作之间设置, 而不是用于整个请求的传输。如果代理服务器在此时间内未接收到任何内容, 则连接将关闭。

## proxy\_set\_body

语法	<code>proxy_set_body value;</code>
默认值	—
上下文	http, server, location

允许重新定义传递给代理服务器的请求主体。值可以包含文本、变量及其组合。

## proxy\_set\_header

语法	<code>proxy_set_header field value;</code>
默认值	<code>proxy_set_header Host \$proxy_host;</code>
上下文	http, server, location

允许重新定义或添加字段到请求头传递给代理服务器。*value* 可以包含文本、变量及其组合。这些指令仅在当前级别未定义 `proxy_set_header` 指令时从前一个配置级别继承。默认情况下, 仅重新定义两个字段:

```
proxy_set_header Host      $proxy_host;  
proxy_set_header Connection close;
```

如果启用了缓存, 原始请求中的头字段”If-Modified-Since”、”If-Unmodified-Since”、”If-None-Match”、”If-Match”、”Range” 和”If-Range” 不会传递给代理服务器。

可以像这样传递未更改的”Host” 请求头字段:

```
proxy_set_header Host      $http_host;
```

然而, 如果在客户端请求头中不存在此字段, 则不会传递任何内容。在这种情况下, 最好使用 `$host` 变量 - 其值等于”Host” 请求头字段中的服务器名称, 或者如果该字段不存在, 则为主服务器名称:

```
proxy_set_header Host      $host;
```

此外, 还可以与代理服务器的端口一起传递服务器名称:

```
proxy_set_header Host      $host:$proxy_port;
```

如果头字段的值为空字符串, 则不会将该字段传递给代理服务器:

```
proxy_set_header Accept-Encoding "";
```

### proxy\_socket\_keepalive

语法	proxy_socket_keepalive on   off;
默认值	proxy_socket_keepalive off;
上下文	http, server, location

配置用于代理服务器的传出连接的“TCP keepalive”行为。

""	默认情况下, 操作系统的设置对套接字生效。
on	为套接字打开 <code>SO_KEEPALIVE</code> 套接字选项。

### proxy\_ssl\_certificate

语法	proxy_ssl_certificate file [file];
默认值	—
上下文	http, server, location

指定用于对代理的 HTTPS 服务器进行身份验证的 PEM 格式证书文件。文件名中可以使用变量。

Added in version 1.2.0.

当 `proxy_ssl_nTLS` 启用时, 指令接受两个参数而不是一个, 分别是证书的签名和加密部分:

```
location /proxy {
    proxy_ssl_nTLS on;

    proxy_ssl_certificate    sign.crt enc.crt;
    proxy_ssl_certificate_key sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass https://backend:443;
}
```



### proxy\_ssl\_certificate\_key

语法	<code>proxy_ssl_certificate_key file [file];</code>
默认值	—
上下文	http, server, location

指定用于对代理的 HTTPS 服务器进行身份验证的 PEM 格式密钥文件。

可以指定“engine:name:id”代替文件，从 OpenSSL 引擎名中加载指定 id 的密钥。文件名中可以使用变量。

Added in version 1.2.0.

当`proxy_ssl_nTLS`启用时，指令接受两个参数而不是一个：密钥的签名和加密部分：

```
location /proxy {
    proxy_ssl_nTLS on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDSA-SM2-WITH-SM4-SM3:RSA";

    proxy_pass https://backend:443;
}
```

### proxy\_ssl\_ciphers

语法	<code>proxy_ssl_ciphers ciphers;</code>
默认值	<code>proxy_ssl_ciphers DEFAULT;</code>
上下文	http, server, location

指定对代理的 HTTPS 服务器的请求启用的密码。密码以 OpenSSL 库理解的格式指定。

完整列表可以使用“`openssl ciphers`”命令查看。

### proxy\_ssl\_conf\_command

语法	<code>proxy_ssl_conf_command name value;</code>
默认值	—
上下文	http, server, location

在与代理的 HTTPS 服务器建立连接时设置任意 OpenSSL 配置命令。

#### 重要

使用 OpenSSL 1.0.2 或更高版本时支持该指令。

可以在同一级别指定多个 `proxy_ssl_conf_command` 指令。只有在当前级别没有定义 `proxy_ssl_conf_command` 指令时, 这些指令才会从上一级配置继承。

#### 小心

请注意, 直接配置 OpenSSL 可能导致意外行为。

### proxy\_ssl\_crl

语法	<code>proxy_ssl_crl file;</code>
默认值	—
上下文	http, server, location

指定用于验证代理的 HTTPS 服务器证书的 PEM 格式撤销证书 (CRL) 文件。

### proxy\_ssl\_name

语法	<code>proxy_ssl_name name;</code>
默认值	<code>proxy_ssl_name \$proxy_host;</code>
上下文	http, server, location

允许覆盖用于验证代理的 HTTPS 服务器证书的服务器名称, 并在与代理的 HTTPS 服务器建立连接时通过 SNI 传递。

默认情况下, 使用 `proxy_pass` URL 的主机部分。

## proxy\_ssl\_ntls

Added in version 1.2.0.

语法	proxy_ssl_ntls on   off;
默认值	proxy_ssl_ntls off;
上下文	http, server

启用使用 `TongSuo` 库的客户端 NTLS 支持。

```
location /proxy {
    proxy_ssl_ntls on;

    proxy_ssl_certificate    sign.crt enc.crt;
    proxy_ssl_certificate_key sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass https://backend:443;
}
```

### 重要

使用 `--with-ntls` 构建选项构建代理并链接到启用 NTLS 的 SSL 库

```
./configure --with-openssl=../Tongsuo-8.3.0 \  
            --with-openssl-opt=enable-ntls \  
            --with-ntls
```

## proxy\_ssl\_password\_file

语法	proxy_ssl_password_file <i>file</i> ;
默认值	—
上下文	http, server, location

指定 `密钥` 的密码短语文件, 每个密码短语在单独的行上指定。在加载密钥时依次尝试密码短语。

### proxy\_ssl\_protocols

语法	<code>proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];</code>
默认值	<code>proxy_ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;</code>
上下文	<code>http, server, location</code>

在 1.2.0 版本发生变更: TLSv1.3 参数已添加到默认集合。

启用对代理的 HTTPS 服务器请求的指定协议。

### proxy\_ssl\_server\_name

语法	<code>proxy_ssl_server_name on   off;</code>
默认值	<code>proxy_ssl_server_name off;</code>
上下文	<code>http, server, location</code>

启用或禁用通过 Server Name Indication TLS 扩展 (SNI, RFC 6066) 传递由 `proxy_ssl_name` 指令设置的服务器名称在与代理的 HTTPS 服务器建立连接时。

### proxy\_ssl\_session\_reuse

- - 语法
  - `proxy_ssl_session_reuse on | off;`
- - 默认值
  - `proxy_ssl_session_reuse on;`
- - 上下文
  - `http, server, location`

确定是否可以在与代理服务器交互时重用 SSL 会话。如果日志中出现“`SSL3_GET_FINISHED:digest check failed`”错误, 请尝试禁用会话重用。

### proxy\_ssl\_trusted\_certificate

语法	<code>proxy_ssl_trusted_certificate file;</code>
默认值	—
上下文	<code>http, server, location</code>

指定用于验证代理的 HTTPS 服务器证书的 PEM 格式受信任 CA 证书文件。

### proxy\_ssl\_verify

语法	<code>proxy_ssl_verify on   off;</code>
默认值	<code>proxy_ssl_verify off;</code>
上下文	http, server, location

启用或禁用对代理的 HTTPS 服务器证书的验证。

### proxy\_ssl\_verify\_depth

语法	<code>proxy_ssl_verify_depth number;</code>
默认值	<code>proxy_ssl_verify_depth 1;</code>
上下文	http, server, location

设置代理的 HTTPS 服务器证书链的验证深度。

### proxy\_store

语法	<code>proxy_store on   off   string;</code>
默认值	<code>proxy_store off;</code>
上下文	http, server, location

启用将文件保存到磁盘。

on	保存文件路径与指令 <i>alias</i> 或 <i>root</i> 对应
off	禁用文件保存

可以使用带有变量的 `string` 明确设置文件名：

```
proxy_store /data/www$original_uri;
```

文件的修改时间根据收到的“Last-Modified”响应头字段设置。响应首先被写入临时文件，然后文件被重命名。临时文件和持久存储可以放在不同的文件系统上。但是，请注意，在这种情况下，文件是在两个文件系统之间复制，而不是廉价的重命名操作。因此，建议在任何给定 *location* 中，保存的文件和由 *proxy\_temp\_path* 指令设置的持有临时文件的目录放在同一个文件系统上。

此指令可用于创建静态不变文件的本地副本，例如：

```
location /images/ {  
    root /data/www;
```

```
    error_page      404 = /fetch$uri;
}

location /fetch/ {
    internal;

    proxy_pass      http://backend/;
    proxy_store     on;
    proxy_store_access user:rw group:rw all:r;
    proxy_temp_path /data/temp;

    alias           /data/www/;
}
```

或者像这样:

```
location /images/ {
    root          /data/www;
    error_page    404 = @fetch;
}

location @fetch {
    internal;

    proxy_pass    http://backend;
    proxy_store   on;
    proxy_store_access user:rw group:rw all:r;
    proxy_temp_path /data/temp;

    root         /data/www;
}
```

### proxy\_store\_access

语法	<code>proxy_store_access users:permissions ...;</code>
默认值	<code>proxy_store_access user:rw;</code>
上下文	http, server, location

设置新创建文件和目录的访问权限, 例如:

```
proxy_store_access user:rw group:rw all:r;
```

如果指定了 `group` 或 `all` 访问权限, 则可以省略用户权限:

```
proxy_store_access group:rw all:r;
```

### proxy\_temp\_file\_write\_size

语法	<code>proxy_temp_file_write_size size;</code>
默认值	<code>proxy_temp_file_write_size 8k 16k;</code>
上下文	http, server, location

限制一次写入临时文件的数据大小, 当启用将代理服务器的响应缓冲到临时文件时。默认情况下, 大小由 `proxy_buffer_size` 和 `proxy_buffers` 指令设置的两个缓冲区限制。临时文件的最大大小由 `proxy_max_temp_file_size` 指令设置。

### proxy\_temp\_path

语法	<code>proxy_temp_path path [level1 [level2 [level3]]];</code>
默认值	<code>proxy_temp_path proxy_temp;</code> (路径依赖于 <code>--http-proxy-temp-path</code> 构建选项)
上下文	http, server, location

定义存储从代理服务器接收的数据的临时文件的目录。可以在指定目录下使用最多三级子目录层次结构。例如, 在以下配置中

```
proxy_temp_path /spool/angie/proxy_temp 1 2;
```

一个临时文件可能如下所示:

```
/spool/angie/proxy_temp/7/45/00000123457
```

另请参见 `proxy_cache_path` 指令的 `use_temp_path` 参数。

### 内置变量

`http_proxy` 模块支持内置变量, 可以使用 `proxy_set_header` 指令来组合头部:

`$proxy_host`

在 `proxy_pass` 指令中指定的代理服务器名称和端口;

`$proxy_port`

在 `proxy_pass` 指令中指定的代理服务器端口, 或协议的默认端口;

`$proxy_add_x_forwarded_for`

带有附加的 `$remote_addr` 变量的“X-Forwarded-For”客户端请求头字段, 以逗号分隔。如果客户端请求头中不存在“X-Forwarded-For”字段, 则 `$proxy_add_x_forwarded_for` 变量等于 `$remote_addr` 变量。

## 随机索引

该模块处理以斜杠字符 (/) 结尾的请求, 并在目录中随机选择一个文件作为索引文件提供服务。该模块在 `http_index` 模块之前处理。

当从源代码构建时, 该模块默认不会被构建; 需要通过 `--with-http_random_index_module` 构建选项来启用。

在从我们的仓库获取的软件包和镜像中, 该模块已包含在构建中。

## 配置示例

```
location / {
    random_index on;
}
```

## 指令

### random\_index

语法	<code>random_index on   off.</code>
默认值	<code>random_index off;</code>
上下文	<code>location</code>

启用或禁用在周围位置中模块的处理。



## RealIP

该模块用于将客户端地址和可选端口更改为指定头字段中发送的地址和端口。

当从源代码构建时, 该模块默认未构建; 需要通过 `--with-http_realip_module` 构建选项进行启用。

在来自我们的仓库的包和镜像中, 该模块已包含在构建中。

## 配置示例

```
set_real_ip_from 192.168.1.0/24;  
set_real_ip_from 192.168.2.1;  
set_real_ip_from 2001:0db8::/32;  
real_ip_header X-Forwarded-For;  
real_ip_recursive on;
```

## 指令

### set\_real\_ip\_from

语法	<code>set_real_ip_from address   CIDR   unix::;</code>
默认	—
上下文	http, server, location

定义已知发送正确替换地址的可信地址。如果指定了特殊值 `unix::`, 则所有 UNIX 域套接字将被信任。可信地址也可以使用主机名指定。

### real\_ip\_header

语法	<code>real_ip_header field   X-Real-IP   X-Forwarded-For   proxy_protocol;</code>
默认	<code>real_ip_header X-Real-IP;</code>
上下文	http, server, location

定义请求头字段, 其值将用于替换客户端地址。

包含可选端口的请求头字段值也用于替换客户端端口。地址和端口应根据 RFC 3986 进行指定。

`proxy_protocol` 参数将客户端地址更改为来自 PROXY 协议头的地址。必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

## real\_ip\_recursive

语法	<code>real_ip_recursive on   off;</code>
默认	<code>real_ip_recursive off;</code>
上下文	<code>http, server, location</code>

如果禁用递归搜索, 则与可信地址之一匹配的原始客户端地址被请求头字段中由 `real_ip_header` 指令定义的最后一个地址替换。如果启用递归搜索, 则与可信地址之一匹配的原始客户端地址被请求头字段中最后一个非可信地址替换。

### 内置变量

`$realip_remote_addr`

保存原始客户端地址

`$realip_remote_port`

保存原始客户端端口

### Referer

该模块用于阻止具有无效“Referer”头字段值的请求访问站点。应注意, 伪造具有适当“Referer”字段值的请求非常容易, 因此该模块的目的不是彻底阻止此类请求, 而是阻止由常规浏览器发送的大量请求流。还应考虑, 即使是有效请求, 常规浏览器也可能不发送“Referer”字段。

### 配置示例

```
valid_referers none blocked server_names
    *.example.com example.* www.example.org/galleries/
    ~\.google\.;

if ($invalid_referer) {
    return 403;
}
```

## 指令

### referer\_hash\_bucket\_size

语法	<code>referer_hash_bucket_size size;</code>
默认值	<code>referer_hash_bucket_size 64;</code>
上下文	server, location

设置有效引用者哈希表的桶大小。有关设置哈希表的详细信息，请参阅单独的 文档。

### referer\_hash\_max\_size

语法	<code>referer_hash_max_size size;</code>
默认值	<code>referer_hash_max_size 2048;</code>
上下文	server, location

设置有效引用者哈希表的最大大小。有关设置哈希表的详细信息，请参阅单独的 文档。

### valid\_referers

语法	<code>valid_referers none   blocked   server_names   string ...;</code>
默认值	—
上下文	server, location

指定将导致内置 `$invalid_referer` 变量被设置为空字符串的“Referer”请求头字段值。否则，变量将被设置为“1”。匹配搜索不区分大小写。

参数可以如下：

<code>none</code>	请求头中缺少“Referer”字段；
<code>blocked</code>	请求头中存在“Referer”字段，但其值已被防火墙或代理服务器删除；此类值是指不以“ <code>http://</code> ”或“ <code>https://</code> ”开头的字符串；
<code>server_names</code>	“Referer”请求头字段包含一个服务器名称；
<code>arbitrary string</code>	定义一个服务器名称和一个可选的 URI 前缀。服务器名称可以在开头或结尾加上“*”。在检查过程中，会忽略“Referer”字段中的服务器端口；
<code>regular expression</code>	第一个符号应为“~”。应注意，表达式将与“ <code>http://</code> ”或“ <code>https://</code> ”之后的文本进行匹配。

示例：

```
valid_referers none blocked server_names
    *.example.com example.* www.example.org/galleries/
    ~\.google\.;
```

## 内置变量

`$invalid_referer`

如果“Referer”请求头字段值被认为是有效的, 则为空字符串, 否则为“1”。

## 重写

该模块用于使用 PCRE 正则表达式更改请求 URI, 返回重定向, 并有条件地选择配置。

`break`、`if`、`return`、`rewrite` 和 `set` 指令按以下顺序处理:

- 在 `server` 级别指定的本模块指令按顺序执行;
- 重复:
  - 根据请求 URI 搜索一个 `location`;
  - 在找到的 `location` 中指定的本模块指令按顺序执行;
  - 如果请求 URI 被重写, 但不超过 10 次, 则循环重复。

## 指令

### break

语法	<code>break;</code>
默认值	—
上下文	<code>server</code> , <code>location</code> , <code>if</code>

停止处理当前的 `http_rewrite` 指令集。

如果在 `location` 内指定了指令, 则请求的进一步处理将在此 `location` 中继续。

示例:

```
if ($slow) {
    limit_rate 10k;
    break;
}
```

## if

语法	<code>if (condition) { ... }</code>
默认值	—
上下文	server, location

指定的条件被评估。如果为真，则括号内指定的本模块指令被执行，并且请求被分配在 `if` 指令中的配置。`if` 指令中的配置继承自上一级配置。

一个条件可以是以下任何一种：

- 一个变量名；如果变量的值为空字符串或“0”，则为假；
- 使用“=”和“!=”操作符将变量与字符串进行比较；
- 使用“~”（区分大小写匹配）和“~\*”（不区分大小写匹配）操作符将变量与正则表达式匹配。正则表达式可以包含捕获，供以后在 `$1..$9` 变量中重用。也可以使用否定操作符“!~”和“!~\*”。如果正则表达式包含“}”或“;”字符，整个表达式应放在单引号或双引号内。
- 使用“-f”和“-f!”操作符检查文件是否存在；
- 使用“-d”和“-d!”操作符检查目录是否存在；
- 使用“-e”和“-e!”操作符检查文件、目录或符号链接是否存在；
- 使用“-x”和“-x!”操作符检查可执行文件。

示例：

```
if ($http_user_agent ~ MSIE) {
    rewrite ^(.*)$ /msie/$1 break;
}

if ($http_cookie ~* "id=(\[^\;]+\)(?:;|$)") {
    set $id $1;
}

if ($request_method = POST) {
    return 405;
}

if ($slow) {
    limit_rate 10k;
}

if ($invalid_referer) {
    return 403;
}
```

**i** 备注

`$invalid_referer` 内置变量的值由 `valid_referers` 指令设置。

**return**

语法	<code>return code [text];</code> <code>return code URL;</code> <code>return URL;</code>
默认值	—
上下文	server, location, if

停止处理并返回指定的 `code` 给客户端。非标准代码 444 关闭连接而不发送响应头。

可以为代码 301、302、303、307 和 308 指定重定向 URL, 或其他代码指定响应体文本。响应体文本和重定向 URL 可以包含变量。作为一种特殊情况, 重定向 URL 可以被指定为此服务器的本地 URI, 在这种情况下, 完整的重定向 URL 根据请求方案 (`$scheme`) 以及 `server_name_in_redirect` 和 `port_in_redirect` 指令形成。

此外, 可以将用于临时重定向的 302 代码 URL 指定为唯一参数。此类参数应以 `http://`、`https://` 或 `$scheme` 字符串开头。URL 可以包含变量。

另见 `error_page` 指令。

**rewrite**

语法	<code>rewrite regex replacement [flag];</code>
默认值	—
上下文	server, location, if

如果指定的正则表达式匹配请求 URI, 则根据 `replacement` 字符串更改 URI。 `rewrite` 指令按其在配置文件中的出现顺序依次执行。可以使用 `flags` 终止进一步的指令处理。如果 `replacement` 字符串以 `http://`、`https://` 或 `$scheme` 开头, 处理将停止并返回重定向给客户端。

可选的 `flag` 参数可以是以下之一:

<code>last</code>	停止处理当前的 <code>http_rewrite</code> 指令集, 并开始搜索与更改后的 URI 匹配的新 <code>location</code> ;
<code>break</code>	停止处理当前的 <code>http_rewrite</code> 指令集, 如同 <code>break</code> 指令;
<code>redirect</code>	返回带有 302 代码的临时重定向; 如果 <code>replacement</code> 字符串不以 <code>http://</code> 、 <code>https://</code> 或 <code>\$scheme</code> 开头时使用;
<code>permanent</code>	返回带有 301 代码的永久重定向。

完整的重定向 URL 根据请求方案 ( $\$scheme$ ) 以及 `server_name_in_redirect` 和 `port_in_redirect` 指令形成。

示例:

```
server {
#   ...
    rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 last;
    rewrite ^(/download/.*)/audio/(.*)\..*$ $1/mp3/$2.ra last;
    return 403;
#   ...
}
```

但如果这些指令放在 `/download/` 位置内部, 则 `last` 标志应替换为 `break`, 否则将进行 10 个循环并返回 500 错误:

```
location /download/ {
    rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 break;
    rewrite ^(/download/.*)/audio/(.*)\..*$ $1/mp3/$2.ra break;
    return 403;
}
```

如果 `replacement` 字符串包含新的请求参数, 则以前的请求参数会附加在它们之后。如果不希望这样, 可以在替换字符串末尾放置一个问号以避免其附加, 例如:

```
rewrite ^/users/(.*)$ /show?user=$1? last;
```

如果正则表达式包含 `”` 或 `;` 字符, 整个表达式应放在单引号或双引号内。

### rewrite\_log

语法	<code>rewrite_log on   off;</code>
默认值	<code>rewrite_log off;</code>
上下文	http, server, location, if

启用或禁用将 `http_rewrite` 模块指令处理结果记录到 `error_log` 中的 `notice` 级别。

### set

语法	<code>set \$variable value;</code>
默认值	—
上下文	server, location, if

为指定变量设置值。该值可以包含文本、变量及其组合。

## uninitialized\_variable\_warn

语法	uninitialized_variable_warn on   off;
默认值	uninitialized_variable_warn on;
上下文	http, server, location, if

控制是否记录未初始化变量的警告。

### 内部实现

`http_rewrite` 模块指令在配置阶段编译为内部指令, 并在请求处理期间解释执行。解释器是一个简单的虚拟堆栈机。

例如, 指令

```
location /download/ {
    if ($forbidden) {
        return 403;
    }

    if ($slow) {
        limit_rate 10k;
    }

    rewrite ^/(download/.*)/media/(.*)\..*$ /$1/mp3/$2.mp3 break;
}
```

将被翻译为这些指令:

```
variable $forbidden
check against zero
    return 403
    end of code
variable $slow
check against zero
match of regular expression
copy "/"
copy $1
copy "/mp3/"
copy $2
copy ".mp3"
end of regular expression
end of code
```

注意, 上述指令中没有关于 `limit_rate` 的指令, 因为它与 `http_rewrite` 模块无关。为 `if` 块创建了一个单独的配置。如果条件为真, 请求被分配此配置, 其中 `limit_rate` 等于 10k。



指令

```
rewrite ^/(download/.*)/media/(.*)\..*$ /$1/mp3/$2.mp3 break;
```

如果正则表达式中的第一个斜杠放在括号内, 可以通过一个指令将其缩小:

```
rewrite ^(/download/.*)/media/(.*)\..*$ $1/mp3/$2.mp3 break;
```

相应的指令将如下所示:

```
match of regular expression
copy $1
copy "/mp3/"
copy $2
copy ".mp3"
end of regular expression
end of code
```

## SCGI

允许将请求传递给 SCGI 服务器。

### 配置示例

```
location / {
    include scgi_params;
    scgi_pass localhost:9000;
}
```

指令

### scgi\_bind

语法	<code>scgi_bind address [transparent]   off;</code>
默认	—
上下文	http, server, location

使到 SCGI 服务器的外发连接源自指定的本地 IP 地址, 端口可选。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 `scgi_bind` 指令的效果, 允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 SCGI 服务器的外发连接源自非本地 IP 地址, 例如, 来自客户端的真实 IP 地址:

```
scgi_bind $remote_addr transparent;
```

为了使此参数生效, 通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上不需要这样做, 因为如果指定了 `transparent` 参数, 工作进程将从主进程继承 `CAP_NET_RAW` 能力。

### 重要

必须配置内核路由表以拦截来自 SCGI 服务器的网络流量。

## scgi\_buffer\_size

语法	<code>scgi_buffer_size size;</code>
默认	<code>scgi_buffer_size 4k 8k;</code>
上下文	http, server, location

设置用于读取从 SCGI 服务器接收到的响应第一部分的缓冲区大小。此部分通常包含一个小的响应头。默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 具体取决于平台。可以将其设置得更小。

## scgi\_buffering

语法	<code>scgi_buffering size;</code>
默认	<code>scgi_buffering on;</code>
上下文	http, server, location

启用或禁用来自 SCGI 服务器的响应缓冲。

on	Angie 尽快从 SCGI 服务器接收响应, 将其保存到由 <code>scgi_buffer_size</code> 和 <code>scgi_buffers</code> 指令设置的缓冲区中。如果整个响应无法放入内存中, 则可以将其一部分保存到磁盘上的临时文件中。写入临时文件由 <code>scgi_max_temp_file_size</code> 和 <code>scgi_temp_file_write_size</code> 指令控制。
off	响应同步传递给客户端, 立即在接收到时传递。Angie 不会尝试从 SCGI 服务器读取整个响应。Angie 从服务器一次可以接收的数据的最大大小由 <code>scgi_buffer_size</code> 指令设置。

缓冲还可以通过在 "X-Accel-Buffering" 响应头字段中传递 "yes" 或 "no" 来启用或禁用。此功能可以使用 `scgi_ignore_headers` 指令禁用。

### scgi\_buffers

语法	<code>scgi_buffers number size;</code>
默认	<code>scgi_buffers 8 4k   8k;</code>
上下文	http, server, location

设置用于从 SCGI 服务器读取响应的缓冲区的数量和大小, 针对单个连接。

默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 具体取决于平台。

### scgi\_busy\_buffers\_size

语法	<code>scgi_busy_buffers_size size;</code>
默认	<code>scgi_busy_buffers_size 8k   16k;</code>
上下文	http, server, location

当启用来自 SCGI 服务器的响应缓冲时, 限制在响应未完全读取的情况下, 可以忙于向客户端发送响应的缓冲区的总大小。同时, 剩余的缓冲区可以用于读取响应, 如果需要, 可以将响应的一部分缓冲到临时文件中。

默认情况下, 大小由 `scgi_buffer_size` 和 `scgi_buffers` 指令设置的两个缓冲区的大小限制。

### scgi\_cache

语法	<code>scgi_cache zone   off;</code>
默认	<code>scgi_cache off;</code>
上下文	http, server, location

定义用于缓存的共享内存区域。相同的区域可以在多个地方使用。参数值可以包含变量。

off	禁用从先前配置级别继承的缓存。
-----	-----------------

### scgi\_cache\_background\_update

语法	<code>scgi_cache_background_update on   off;</code>
默认	<code>scgi_cache_background_update off;</code>
上下文	http, server, location

允许启动后台子请求来更新过期的缓存项, 同时向客户端返回陈旧的缓存响应。

#### ⚠ 注意

请注意, 当正在更新缓存响应时, 需要允许使用陈旧的缓存响应。

### scgi\_cache\_bypass

语法	<code>scgi_cache_bypass ...;</code>
默认	—
上下文	http, server, location

定义在何种条件下响应不会从缓存中获取。如果字符串参数的至少一个值不为空且不等于”0”, 则响应不会从缓存中获取:

```
scgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;  
scgi_cache_bypass $http_pragma $http_authorization;
```

可以与 `scgi_no_cache` 指令一起使用。

### scgi\_cache\_key

语法	<code>scgi_cache_key string;</code>
默认	—
上下文	http, server, location

定义缓存的键, 例如

```
scgi_cache_key localhost:9000$request_uri;
```

### scgi\_cache\_lock

语法	<code>scgi_cache_lock on   off;</code>
默认	<code>scgi_cache_lock off;</code>
上下文	http, server, location

启用时, 同一时间只允许一个请求填充一个新的缓存元素, 该元素根据 `scgi_cache_key` 指令通过将请求传递给 SCGI 服务器来识别。相同缓存元素的其他请求将等待响应出现在缓存中或该元素的缓存锁释放, 最多等待 `scgi_cache_lock_timeout` 指令设置的时间。

### scgi\_cache\_lock\_age

语法	<code>scgi_cache_lock_age time;</code>
默认	<code>scgi_cache_lock_age 5s;</code>
上下文	http, server, location

如果传递给 SCGI 服务器以填充新缓存元素的最后一个请求在指定时间内未完成, 则可以再传递一个请求给 SCGI 服务器。

### scgi\_cache\_lock\_timeout

语法	<code>scgi_cache_lock_timeout time;</code>
默认	<code>scgi_cache_lock_timeout 5s;</code>
上下文	http, server, location

为 `scgi_cache_lock` 设置超时时间。当时间到期时, 请求将被传递给 SCGI 服务器, 但响应不会被缓存。

### scgi\_cache\_max\_range\_offset

语法	<code>scgi_cache_max_range_offset number;</code>
默认	—
上下文	http, server, location

为字节范围请求设置偏移量 (以字节为单位)。如果范围超出偏移量, 范围请求将被传递给 SCGI 服务器, 并且响应不会被缓存。

### scgi\_cache\_methods

语法	<code>scgi_cache_methods GET   HEAD   POST ...;</code>
默认	<code>scgi_cache_methods GET HEAD;</code>
上下文	<code>http, server, location</code>

如果客户端请求方法列在此指令中, 则响应将被缓存。”GET”和”HEAD”方法总是会添加到列表中, 尽管建议明确指定它们。另请参阅 `scgi_no_cache` 指令。

### scgi\_cache\_min\_uses

语法	<code>scgi_cache_min_uses number;</code>
默认	<code>scgi_cache_min_uses 1;</code>
上下文	<code>http, server, location</code>

设置在响应被缓存之前的请求次数。

### scgi\_cache\_path

语法	<code>scgi_cache_path path [levels=levels] [use_temp_path=on   off] keys_zone=name:size [inactive=time] [max_size=size] [min_free=size] [manager_files=number] [manager_sleep=time] [manager_threshold=time] [loader_files=number] [loader_sleep=time] [loader_threshold=time];</code>
默认	—
上下文	<code>http</code>

设置缓存的路径和其他参数。缓存数据存储在文件中。缓存中的文件名是将 MD5 函数应用于缓存键的结果。

`levels` 参数定义了缓存的层级: 从 1 到 3, 每个层级接受 1 或 2 的值。例如, 在以下配置中:

```
scgi_cache_path /data/angie/cache levels=1:2 keys_zone=one:10m;
```

缓存中的文件名称将如下所示:

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存的响应首先写入一个临时文件, 然后文件会被重命名。临时文件和缓存可以放置在不同的文件系统中。然而, 需要注意的是, 在这种情况下, 文件会在两个文件系统之间复制, 而不是进行简单的重命名操作。因此, 建议将任何给定位置的缓存和保存临时文件的目录放在同一个文件系统中。

临时文件的目录是根据 `use_temp_path` 参数设置的。

on	如果省略此参数或设置为 on, 将使用 <code>scgi_temp_path</code> 指令为给定 <code>location</code> 设置的目录。
off	临时文件将直接放在缓存目录中。

此外, 所有活动的键和关于数据的信息都存储在一个共享内存区中, 其名称和大小由 `keys_zone` 参数配置。一个兆字节的区域可以存储大约 8000 个键。

在 `inactive` 参数指定的时间内没有被访问的缓存数据会被移除, 无论其新鲜度如何。

默认情况下, `inactive` 设置为 10 分钟。

一个特殊的 **cache manager** 进程监视最大缓存大小和文件系统上的最小可用空间, 当大小超过或没有足够的可用空间时, 它会删除最近最少使用的数据。数据是在迭代中移除的。

<code>max_size</code>	最大缓存大小
<code>min_free</code>	文件系统上的最小可用空间
<code>manager_files</code>	限制一次迭代中要删除的项目数量 默认情况下, 100
<code>manager_threshold</code>	限制一次迭代的持续时间 默认情况下, 200 毫秒
<code>manager_sleep</code>	配置迭代之间的暂停 默认情况下, 50 毫秒

Angie 启动一分钟后, 特殊的 **cache loader** 进程被激活。它将之前存储在文件系统上的缓存数据加载到缓存区中。加载也是在迭代中进行的。

<code>loader_files</code>	限制一次迭代中要加载的项目数量 默认情况下, 100
<code>loader_threshold</code>	限制一次迭代的持续时间 默认情况下, 200 毫秒
<code>loader_sleep</code>	配置迭代之间的暂停 默认情况下, 50 毫秒

### scgi\_cache\_revalidate

语法	<code>scgi_cache_revalidate on   off;</code>
默认	<code>scgi_cache_revalidate off;</code>
上下文	http, server, location

启用使用条件请求 (如 "If-Modified-Since" 和 "If-None-Match" 头字段) 对过期缓存项的重新验证。

## scgi\_cache\_use\_stale

语法	scgi_cache_use_stale error   timeout   invalid_header   updating   http_500   http_502   http_503   http_504   http_403   http_404   http_429   off ...;
默认	scgi_cache_use_stale off;
上下文	http, server, location

确定在与 SCGI 服务器通信时可以在哪些情况下使用过时的缓存响应。该指令的参数与 `scgi_next_upstream` 指令的参数匹配。

error	允许在无法选择处理请求的 SCGI 服务器时使用过时的缓存响应。
updating	额外参数, 允许在缓存响应正在更新时使用过时的缓存响应。这可以在更新缓存数据时最大限度地减少对 SCGI 服务器的访问次数。

使用过时的缓存响应还可以在响应头中直接启用, 指定响应过时后多少秒内继续使用:

- `stale-while-revalidate` 扩展"Cache-Control" 头字段允许在响应正在更新时使用过时的缓存响应。
- `stale-if-error` 扩展"Cache-Control" 头字段允许在发生错误时使用过时的缓存响应。

### 备注

这比使用指令参数的优先级低。

要在填充新缓存元素时最大限度地减少对 SCGI 服务器的访问次数, 可以使用 `scgi_cache_lock` 指令。

## scgi\_cache\_valid

Syntax	scgi_cache_valid [code ...] time;
Default	—
Context	http, server, location

设置不同响应代码的缓存时间。例如, 以下指令

```
scgi_cache_valid 200 302 10m;  
scgi_cache_valid 404 1m;
```

为代码 200 和 302 的响应设置 10 分钟的缓存时间, 为代码 404 的响应设置 1 分钟。

如果只指定缓存时间



```
scgi_cache_valid 5m;
```

那么只有 200、301 和 302 的响应会被缓存。

此外, 可以指定 `any` 参数以缓存任何响应:

```
scgi_cache_valid 200 302 10m;  
scgi_cache_valid 301      1h;  
scgi_cache_valid any      1m;
```

#### **i** 备注

缓存的参数也可以直接在响应头中设置。这比使用指令设置缓存时间的优先级高。

- "X-Accel-Expires" 头字段以秒设置响应的缓存时间。值为零将禁用响应的缓存。如果值以 @ 前缀开头, 则设置一个自 Epoch 以来的绝对时间, 以秒为单位, 直到该时间为止可以缓存响应。
- 如果头部不包含 "X-Accel-Expires" 字段, 缓存参数可以在 "Expires" 或 "Cache-Control" 头字段中设置。
- 如果头部包含 "Set-Cookie" 字段, 则该响应将不被缓存。
- 如果头部包含特殊值 "\*" 的 "Vary" 字段, 则该响应将不被缓存。如果头部包含其他值的 "Vary" 字段, 该响应将根据相应的请求头字段进行缓存。

可以使用 `scgi_ignore_headers` 指令禁用对这些响应头字段之一或多个的处理。

### scgi\_connect\_timeout

Syntax	<code>scgi_connect_timeout time;</code>
Default	<code>scgi_connect_timeout 60s;</code>
Context	http, server, location

定义与 SCGI 服务器建立连接的超时时间。需要注意的是, 此超时时间通常不能超过 75 秒。

### scgi\_connection\_drop

Syntax	<code>scgi_connection_drop time   on   off;</code>
Default	<code>scgi_connection_drop off;</code>
Context	http, server, location

启用在代理服务器被从组中移除或被 `eresolve` 进程或 `API` 命令 `DELETE` 标记为永久不可用后终止所有连接。

当为客户端或代理服务器处理下一个读取或写入事件时, 连接被终止。

设置 *time* 启用连接终止 超时; 设置为 *on* 时, 连接将立即断开。

### scgi\_force\_ranges

Syntax	<code>scgi_force_ranges off;</code>
Default	<code>scgi_force_ranges off;</code>
Context	http, server, location

为来自 SCGI 服务器的缓存和未缓存响应启用字节范围支持, 无论这些响应中是否包含“Accept-Ranges”字段。

### scgi\_hide\_header

Syntax	<code>scgi_hide_header field;</code>
Default	—
Context	http, server, location

默认情况下, Angie 不会将来自 SCGI 服务器响应的“Status”和“X-Accel-...”头字段传递给客户端。*scgi\_hide\_header* 指令设置不传递的额外字段。相反, 如果需要允许字段传递, 可以使用 *scgi\_pass\_header* 指令。

### scgi\_ignore\_client\_abort

Syntax	<code>scgi_ignore_client_abort on   off;</code>
Default	<code>scgi_ignore_client_abort off;</code>
Context	http, server, location

确定在客户端关闭连接而不等待响应时是否应关闭与 SCGI 服务器的连接。

### scgi\_ignore\_headers

Syntax	<code>scgi_ignore_headers field ...;</code>
Default	—
Context	http, server, location

禁用来自 SCGI 服务器的某些响应头字段的处理。可以忽略以下字段: "X-Accel-Redirect"、"X-Accel-Expires"、"X-Accel-Limit-Rate"、"X-Accel-Buffering"、"X-Accel-Charset"、"Expires"、"Cache-Control"、"Set-Cookie" 和 "Vary"。

如果没有禁用, 这些头字段的处理将产生如下效果:

- "X-Accel-Expires"、"Expires"、"Cache-Control"、"Set-Cookie" 和 "Vary" 设置响应缓存 的参数;
- "X-Accel-Redirect" 执行到指定 URI 的内部 重定向;
- "X-Accel-Limit-Rate" 设置响应传输到客户端的速率限制;
- "X-Accel-Buffering" 启用或禁用响应的缓冲;
- "X-Accel-Charset" 设置响应的期望字符集。

### scgi\_intercept\_errors

Syntax	<code>scgi_intercept_errors on   off;</code>
Default	<code>scgi_intercept_errors off;</code>
Context	http, server, location

决定是否将状态码大于或等于 300 的 SCGI 响应传递给客户端, 还是拦截并重定向到 `Angie`, 以使用 `error_page` 指令进行处理。

### scgi\_limit\_rate

Syntax	<code>scgi_limit_rate rate;</code>
Default	<code>scgi_limit_rate 0;</code>
Context	http, server, location

限制从 SCGI 服务器读取响应的速度。 `rate` 以每秒字节数指定, 可以包含变量。

0	禁用速率限制
---	--------

#### 备注

限制是针对每个请求设置的, 因此如果 `Angie` 同时打开两个到 SCGI 服务器的连接, 总体速率将是指定限制的两倍。仅当启用了从 SCGI 服务器的响应的 `buffering` 时, 限制才有效。

### scgi\_max\_temp\_file\_size

Syntax	<code>scgi_max_temp_file_size size;</code>
Default	<code>scgi_max_temp_file_size 1024m;</code>
Context	http, server, location

当启用了从 SCGI 服务器的响应的 *buffering*, 并且整个响应不适合由 *scgi\_buffer\_size* 和 *scgi\_buffers* 指令设置的缓冲区时, 响应的一部分可以保存到临时文件中。此指令设置临时文件的最大大小。一次写入临时文件的数据大小由 *scgi\_temp\_file\_write\_size* 指令设置。

0	禁用响应到临时文件的缓冲
---	--------------

#### 备注

此限制不适用于将被缓存或存储在磁盘上的响应。

### scgi\_next\_upstream

Syntax	<code>scgi_next_upstream error   timeout   invalid_header   http_500   http_503   http_403   http_404   http_429   non_idempotent   off ...;</code>
Default	<code>scgi_next_upstream error timeout;</code>
Context	http, server, location

指定在哪些情况下请求应传递到 *upstream pool* 中的下一个服务器:

<code>error</code>	在与服务器建立连接、传递请求或读取响应头时发生错误;
<code>timeout</code>	在与服务器建立连接、传递请求或读取响应头时发生超时;
<code>invalid_header</code>	服务器返回空的或无效的响应;
<code>http_500</code>	服务器返回代码为 500 的响应;
<code>http_503</code>	服务器返回代码为 503 的响应;
<code>http_403</code>	服务器返回代码为 403 的响应;
<code>http_404</code>	服务器返回代码为 404 的响应;
<code>http_429</code>	服务器返回代码为 429 的响应;
<code>non_idempotent</code>	通常, 使用 <b>非幂等</b> 方法 (POST, LOCK, PATCH) 的请求不会传递到下一个服务器, 如果请求已发送到上游服务器; 启用此选项明确允许重试此类请求;
<code>off</code>	禁止将请求传递到下一个服务器。

**i 备注**

需要注意的是, 只有在尚未向客户端发送任何内容的情况下, 才有可能将请求传递到下一个服务器。也就是说, 如果在传输响应的中途发生错误或超时, 则无法修复。

该指令还定义了与服务器通信的[尝试失败](#) 的标准。

error	总是被视为不成功的尝试, 即使它们没有在指令中指定
timeout	
invalid_header	
http_500	只有在指令中指定时才被视为不成功的尝试
http_503	
http_429	
http_403	从不被视为不成功的尝试
http_404	

将请求传递到下一个服务器可以通过[尝试次数](#) 和[时间](#) 进行限制。

### scgi\_next\_upstream\_timeout

Syntax	<code>scgi_next_upstream_timeout <i>time</i>;</code>
Default	<code>scgi_next_upstream_timeout 0;</code>
Context	http, server, location

限制在将请求传递到下一个 服务器的时间。

0	关闭此限制
---	-------

### scgi\_next\_upstream\_tries

Syntax	<code>scgi_next_upstream_tries <i>number</i>;</code>
Default	<code>scgi_next_upstream_tries 0;</code>
Context	http, server, location

限制将请求传递到下一个 服务器的尝试次数。

0	关闭此限制
---	-------

### scgi\_no\_cache

Syntax	<code>scgi_no_cache string ...;</code>
Default	—
Context	http, server, location

定义在何种情况下响应不会被保存到缓存中。如果字符串参数中的至少一个值不为空且不等于“0”，则响应将不会被保存：

```
scgi_no_cache $cookie_nocache $arg_nocache$arg_comment;  
scgi_no_cache $http_pragma $http_authorization;
```

可以与 `scgi_cache_bypass` 指令一起使用。

### scgi\_param

Syntax	<code>scgi_param parameter value [if_not_empty];</code>
Default	—
Context	http, server, location

设置应传递给 SCGI 服务器的参数。该值可以包含文本、变量及其组合。仅当当前级别未定义 `scgi_param` 指令时，这些指令才从上一级配置继承。

标准 ‘CGI 环境变量 <https://datatracker.ietf.org/doc/html/rfc3875#section-4.1>’ 应作为 SCGI 头提供，请参阅分发中提供的 `scgi_params` 文件：

```
location / {  
    include scgi_params;  
#    ...  
}
```

如果指令中指定了 `if_not_empty`，则只有在参数值不为空时，该参数才会传递给服务器：

```
scgi_param HTTPS $https if_not_empty;
```

### scgi\_pass

Syntax	<code>scgi_pass uri;</code>
Default	—
Context	location, if in location, limit_except

设置 SCGI 服务器的地址。地址可以指定为域名或 IP 地址，以及可选的端口：

```
scgi_pass localhost:9000;
```

或作为用冒号括起来的 UNIX 域套接字路径, 紧跟在 `unix` 之后:

```
scgi_pass unix:/tmp/scgi.socket;
```

如果域名解析为多个地址, 则会轮流使用所有地址。此外, 地址可以指定为服务器组。如果使用组, 则不能为其指定端口; 而是为组内的每台服务器单独指定端口。

参数值可以包含变量。在这种情况下, 如果地址指定为域名, 则会在描述的服务器组中搜索该名称, 如果未找到, 则使用 *resolver* 确定。

### scgi\_pass\_header

Syntax	<code>scgi_pass_header field ...;</code>
Default	—
Context	http, server, location

允许将否则禁用 的头字段从 SCGI 服务器传递到客户端。

### scgi\_pass\_request\_body

Syntax	<code>scgi_pass_request_body on   off;</code>
Default	<code>scgi_pass_request_body on;</code>
Context	http, server, location

指示是否将原始请求体传递给 SCGI 服务器。另请参阅 `scgi_pass_request_headers` 指令。

### scgi\_pass\_request\_headers

语法	<code>scgi_pass_request_headers on   off;</code>
默认值	<code>scgi_pass_request_headers on;</code>
上下文	http, server, location

指示是否将原始请求的头字段传递给 SCGI 服务器。另请参阅 `scgi_pass_request_body` 指令。

### scgi\_read\_timeout

语法	<code>scgi_read_timeout</code> 时间;
默认值	<code>scgi_read_timeout 60s</code> ;
上下文	http, server, location

定义从 SCGI 服务器读取响应的超时时间。超时时间仅在两次连续读取操作之间设置，而不是整个响应的传输时间。如果 SCGI 服务器在此时间内未传输任何内容，则连接将关闭。

### scgi\_request\_buffering

语法	<code>scgi_request_buffering on   off</code> ;
默认值	<code>scgi_request_buffering on</code> ;
上下文	http, server, location

启用或禁用客户端请求体的缓冲。

on	在将请求发送到 SCGI 服务器之前，从客户端读取完整的请求体 <i>client body buffer size</i> 。
off	请求体在接收到时立即发送到 SCGI 服务器。在这种情况下，如果已经开始发送请求体，则无法将请求传递给下一个服务器。

当使用 HTTP/1.1 分块传输编码发送原始请求体时，无论指令值如何，请求体都会被缓冲。

### scgi\_send\_timeout

语法	<code>scgi_send_timeout</code> 时间;
默认值	<code>scgi_send_timeout 60s</code> ;
上下文	http, server, location

设置将请求传输到 SCGI 服务器的超时时间。超时仅在两次连续写操作之间设置，而不是用于整个请求的传输。如果 SCGI 服务器在此时间内没有接收到任何内容，则连接将关闭。



### scgi\_socket\_keepalive

语法	scgi_socket_keepalive on   off;
默认值	scgi_socket_keepalive off;
上下文	http, server, location

配置与 SCGI 服务器的外部连接的“TCP 保持活动”行为。

""	默认情况下, 操作系统的设置在套接字上生效。
on	SO_KEEPALIVE 套接字选项为套接字启用。

### scgi\_store

语法	scgi_store on   off   字符串;
默认值	scgi_store off;
上下文	http, server, location

启用将文件保存到磁盘。

on	保存文件的路径与指令 <i>alias</i> 或 <i>root</i> 相对应
off	禁止保存文件

可以使用带有变量的 字符串显式设置文件名:

```
scgi_store /data/www$original_uri;
```

文件的修改时间根据接收到的“Last-Modified”响应头字段设置。响应首先写入一个临时文件, 然后文件被重命名。临时文件和持久存储可以放在不同的文件系统上。但请注意, 在这种情况下, 文件是在两个文件系统之间复制的, 而不是廉价的重命名操作。因此, 建议在任何给定的 *location* 中, 将保存的文件和由 *scgi\_temp\_path* 指令设置的临时文件目录放在同一个文件系统上。

此指令可用于创建静态不可更改文件的本地副本, 例如:

```
location /images/ {
    root          /data/www;
    error_page    404 = /fetch$uri;
}

location /fetch/ {
    internal;
```

```
scgi_pass          backend:9000;
...

scgi_store         on;
scgi_store_access user:rw group:rw all:r;
scgi_temp_path    /data/temp;

alias              /data/www/;
}
```

### scgi\_store\_access

语法	scgi_store_access 用户: 权限...;
默认值	scgi_store_access user:rw;
上下文	http, server, location

设置新建文件和目录的访问权限, 例如:

```
scgi_store_access user:rw group:rw all:r;
```

如果指定了任何 group 或 all 访问权限, 则可以省略用户权限:

```
scgi_store_access group:rw all:r;
```

### scgi\_temp\_file\_write\_size

语法	scgi_temp_file_write_size 大小;
默认值	scgi_temp_file_write_size 8k 16k;
上下文	http, server, location

限制当启用从 SCGI 服务器到临时文件的响应缓冲时, 一次写入临时文件的数据大小。默认情况下, 大小受由 *scgi\_buffer\_size* 和 *scgi\_buffers* 指令设置的两个缓冲区限制。临时文件的最大大小由 *scgi\_max\_temp\_file\_size* 指令设置。

## scgi\_temp\_path

语法	<code>scgi_temp_path</code> 路径 [ <i>level1</i> [ <i>level2</i> [ <i>level3</i> ]]];
默认值	<code>scgi_temp_path scgi_temp</code> ; (路径取决于 <code>--http-scgi-temp-path</code> 构建选项)
上下文	http, server, location

定义用于存储从 SCGI 服务器接收的数据的临时文件的目录。可以在指定目录下使用最多三级子目录层次结构。例如, 在以下配置中

```
scgi_temp_path /spool/angie/scgi_temp 1 2;
```

临时文件可能如下所示:

```
/spool/angie/scgi_temp/7/45/00000123457
```

另见 `scgi_cache_path` 指令的 `use_temp_path` 参数。

## 安全链接

该模块用于检查请求链接的真实性, 保护资源免受未经授权的访问, 并限制链接的生命周期。

请求链接的真实性通过比较请求中传递的校验和值和为请求计算的值来验证。如果链接有有限的生命周期且时间已过期, 则该链接被视为过时。这些检查的状态在 `$secure_link` 变量中可用。

该模块提供两种替代操作模式。第一种模式通过 `secure_link_secret` 指令启用, 用于检查请求链接的真实性以及保护资源免受未经授权的访问。第二种模式通过 `secure_link` 和 `secure_link_md5` 指令启用, 也用于限制链接的生命周期。

当从源代码构建时, 此模块默认不构建; 应使用 `--with-http_secure_link_module` 构建选项启用。

在来自 我们的仓库的软件包和镜像中, 模块包含在构建中。

## 指令

### secure\_link

语法	<code>secure_link</code> 表达式;
默认值	—
上下文	http, server, location

定义一个包含变量的字符串, 用于从中提取链接的校验和值和生命周期。

表达式中使用的变量通常与请求相关联; 请参阅下面的示例。

从字符串中提取的校验和值与由 `secure_link_md5` 指令定义的表达式的 MD5 哈希值进行比较。

如果校验和不同, 则 `$secure_link` 变量被设置为空字符串。如果校验和相同, 则检查链接的生命周期。

如果链接有有限的使用寿命且时间已过期, 则 `$secure_link` 变量被设置为 0。否则, 它被设置为 1。请求中传递的 MD5 哈希值以 base64url 编码。

如果链接有有限的使用寿命, 则过期时间以自纪元 (Thu, 01 Jan 1970 00:00:00 GMT) 以来的秒数设置。该值在表达式中指定于 MD5 哈希之后, 并以逗号分隔。请求中传递的过期时间可通过 `$secure_link_expires` 变量用于 `secure_link_md5` 指令。如果未指定过期时间, 则链接具有无限的使用寿命。

### secure\_link\_md5

语法	<code>secure_link_md5</code> 表达式;
默认值	—
上下文	http, server, location

定义一个表达式, 其 MD5 哈希值将被计算并与请求中传递的值进行比较。

表达式应包含链接 (资源) 的受保护部分和一个秘密成分。如果链接有有限的使用寿命, 表达式还应包含 `$secure_link_expires`。

为了防止未经授权的访问, 表达式可以包含一些关于客户端的信息, 例如其地址和浏览器版本。

示例:

```
location /s/ {
    secure_link $arg_md5,$arg_expires;
    secure_link_md5 "$secure_link_expires$uri$remote_addr secret";

    if ($secure_link = "") {
        return 403;
    }

    if ($secure_link = "0") {
        return 410;
    }

    # ...
}
```

链接 `"/s/link?md5=__e4Nc3iduzkWRm01TBBNYw&expires=2147483647"` 限制了对客户端 IP 地址为 127.0.0.1 的 `"/s/link"` 的访问。该链接也有有限的使用寿命, 直到 2038 年 1 月 19 日 (GMT)。

在 UNIX 上, 可以通过以下命令获取 md5 请求参数值:

```
echo -n '2147483647/s/link127.0.0.1 secret' | \
    openssl md5 -binary | openssl base64 | tr +/ -_ | tr -d =
```

## secure\_link\_secret

语法	secure_link_secret 单词;
默认值	—
上下文	location

定义一个秘密单词，用于检查请求链接的真实性。

请求链接的完整 URI 如下所示：

```
/prefix/hash/link
```

其中 hash 是为链接和秘密单词连接而计算的 MD5 哈希的十六进制表示，prefix 是没有斜杠的任意字符串。

如果请求链接通过了真实性检查，则 `$secure_link` 变量被设置为从请求 URI 提取的链接。否则，`$secure_link` 变量被设置为空字符串。

示例：

```
location /p/ {
    secure_link_secret secret;

    if ($secure_link = "") {
        return 403;
    }

    rewrite ^ /secure/$secure_link;
}

location /secure/ {
    internal;
}
```

请求 `/p/5e814704a28d9bc1914ff19fa0c4a00a/link` 将被内部重定向到 `/secure/link`。

在 UNIX 上，可以通过以下命令获取此示例的哈希值：

```
echo -n 'linksecret' | openssl md5 -hex
```

## 内置变量

`$secure_link`

链接检查的状态。具体值取决于所选的操作模式。

`$secure_link_expires`

在请求中传递的链接的生命周期；仅用于 `secure_link_md5` 指令。

## 切片

该模块是一个过滤器，将请求拆分为子请求，每个子请求返回特定范围的响应。该过滤器提供了更有效的大响应缓存。

当从源代码构建时，默认情况下不会构建此模块；它需要通过 `--with-http_slice_module` 构建选项启用。

在来自 我们仓库的包和镜像中，该模块已包含在构建中。

## 配置示例

```
location / {
    slice          1m;
    proxy_cache    cache;
    proxy_cache_key $uri$is_args$args$slice_range;
    proxy_set_header Range $slice_range;
    proxy_cache_valid 200 206 1h;
    proxy_pass     http://localhost:8000;
}
```

在此示例中，响应被分割为 1 兆字节可缓存的切片。

## 指令

`slice`

语法	<code>slice size;</code>
默认值	<code>slice 0;</code>
上下文	http, server, location

设置切片的大小。零值将禁用将响应分割为切片。

**警告**

请注意, 值过低可能导致过多的内存使用和打开大量文件。

为了使子请求返回所需的范围, 应该将 `$slice_range` 变量作为 “Range” 请求头字段传递给代理服务器。如果启用了缓存, 应将 `$slice_range` 添加到缓存键中, 并应启用对状态码为 206 的响应的缓存。

**内置变量**

`$slice_range`

以 HTTP 字节范围格式表示的当前切片范围, 例如, `bytes=0-1048575`。

**拆分客户端**

该模块用于 A/B 测试、金丝雀发布或其他需要将一部分客户端路由到一个服务器或配置, 同时将其余部分路由到其他地方的场景。

**配置示例**

```
http {
    split_clients "${remote_addr}AAA" $variant {
        0.5%          .one;
        2.0%          .two;
        *             "";
    }

    server {
        location / {
            index index${variant}.html;
        }
    }
}
```

**指令****split\_clients**

语法	<code>split_clients string \$variable { ... }</code>
默认值	—
上下文	http

通过哈希 `string` 创建一个 `$variable`; `string` 中的变量被替换, 替换结果被哈希, 然后将哈希值映射为 `$variable` 的字符串值。

哈希函数使用 MurmurHash2 (32 位), 其整个值范围 (0 到 4294967295) 按出现顺序映射到桶中; 百分比决定了桶的大小。通配符 (\*) 可以出现在最后; 落在其他桶之外的哈希值映射为其指定的值。

一个示例:

```
split_clients "${remote_addr}AAA" $variant {  
    0.5%          .one;  
    2.0%          .two;  
    *             "";  
}
```

在这里, 插值后的 `$remote_addrAAA` 字符串的哈希值分布如下:

- 值 0 到 21474835 (0.5% 的样本) 产生 `.one`
- 值 21474836 到 107374180 (2% 的样本) 产生 `.two`
- 值 107374181 到 4294967295 (所有其他值) 产生 `""` (一个空字符串)

## SSI

该模块是一个过滤器, 用于处理通过它的响应中的 SSI (服务器端包含) 命令。

### 配置示例

```
location / {  
    ssi on;  
    # ...  
}
```

## 指令

### ssi

Syntax	<code>ssi on   off;</code>
默认	<code>ssi off;</code>
Context	http, server, location, if in location

启用或禁用响应中的 SSI 命令处理。



### ssi\_last\_modified

Syntax	<code>ssi_last_modified on   off;</code>
默认	<code>ssi_last_modified off;</code>
Context	http, server, location

允许在 SSI 处理期间保留原始响应中的“Last-Modified”头字段，以便于响应缓存。

默认情况下，该头字段被删除，因为响应的内容在处理过程中被修改，并且可能包含动态生成的元素或独立于原始响应更改的部分。

### ssi\_min\_file\_chunk

Syntax	<code>ssi_min_file_chunk size;</code>
默认	<code>ssi_min_file_chunk 1k;</code>
Context	http, server, location

设置存储在磁盘上响应部分的最小大小，从该大小开始，使用 *sendfile* 发送它们是有意义的。

### ssi\_silent\_errors

Syntax	<code>ssi_silent_errors on   off;</code>
默认	<code>ssi_silent_errors off;</code>
Context	http, server, location

如果启用，则在 SSI 处理过程中出现错误时，会抑制输出”*[an error occurred while processing the directive]*”字符串。

### ssi\_types

Syntax	<code>ssi_types mime-type ...;</code>
默认	<code>ssi_types text/html;</code>
Context	http, server, location

在响应中启用对指定 MIME 类型的 SSI 命令处理，除了”*text/html*”之外。特殊值”\*”匹配任何 MIME 类型。

## ssi\_value\_length

Syntax	<code>ssi_value_length length;</code>
默认	<code>ssi_value_length 256;</code>
Context	http, server, location

设置 SSI 命令中参数值的最大长度。

## SSI 命令

SSI 命令具有以下通用格式:

```
<!--# command parameter1=value1 parameter2=value2 ... -->
```

支持以下命令:

### block

定义一个可以在 `include` 命令中用作存根的块。该块可以包含其他 SSI 命令。命令具有以下参数:

#### name

块名称。

示例:

```
<!--# block name="one" -->  
stub  
<!--# endblock -->
```

### config

设置 SSI 处理过程中使用的一些参数, 即:

### errmsg

如果 SSI 处理过程中发生错误, 则输出的字符串。默认情况下, 输出以下字符串:

*[an error occurred while processing the directive]*

### timefmt

传递给 `strftime()` 函数用于输出日期和时间的格式字符串。默认情况下, 使用以下格式:

```
"%A, %d-%b-%Y %H:%M:%S %Z"
```

”%s” 格式适用于输出秒数。

### echo

输出变量的值。命令具有以下参数:

#### var

变量名称。

#### encoding

编码方法。可能的值包括 `none`, `url`, 和 `entity`。

默认使用 `entity`。

#### default

一个非标准参数, 设置当变量未定义时输出的字符串。默认情况下, 输出”(`none`)”。命令

```
<!--# echo var="name" default="no" -->
```

替换以下命令序列:

```
<!--# if expr="$name" --><!--# echo var="name" --><!--#  
else -->no<!--# endif -->
```

### if

执行条件包含。支持以下命令:

```
<!--# if expr="..." -->  
...  
<!--# elif expr="..." -->  
...  
<!--# else -->  
...  
<!--# endif -->
```

目前, 仅支持一个嵌套级别。命令具有以下参数:

**expr**

表达式。表达式可以是:

- 变量存在性检查:

```
<!--# if expr="$name" -->
```

- 变量与文本的比较:

```
<!--# if expr="$name = text" -->  
<!--# if expr="$name != text" -->
```

- 变量与正则表达式的比较:

```
<!--# if expr="$name = /text/" -->  
<!--# if expr="$name != /text/" -->
```

如果 *text* 中包含变量, 则替换其值。正则表达式可以包含位置和命名捕获, 稍后可以通过变量使用, 例如:

```
<!--# if expr="$name = /(.)@(P<domain>.+)/" -->  
  <!--# echo var="1" -->  
  <!--# echo var="domain" -->  
<!--# endif -->
```

**include**

将另一个请求的结果包含在响应中。命令具有以下参数:

**file**

指定包含的文件, 例如:

```
<!--# include file="footer.html" -->
```

## virtual

指定包含的请求, 例如:

```
<!--# include virtual="/remote/body.php?argument=value" -->
```

在一个页面上指定的多个请求并由代理或 FastCGI/uwsgi/SCGI/gRPC 服务器处理时并行运行。如果需要顺序处理, 则应使用 *wait* 参数。

## stub

一个非标准参数, 指定如果包含的请求结果为空或请求处理过程中发生错误时将输出的块的名称, 例如:

```
<!--# block name="one" -->&nbsp;  <!--# endblock -->  
<!--# include virtual="/remote/body.php?argument=value" stub="one" -->
```

替换块内容在包含的请求上下文中处理。

## wait

一个非标准参数, 指示在继续 SSI 处理之前等待请求完全完成, 例如:

```
<!--# include virtual="/remote/body.php?argument=value" wait="yes" -->
```

## set

一个非标准参数, 指示将请求处理的成功结果写入指定变量, 例如:

```
<!--# include virtual="/remote/body.php?argument=value" set="one" -->
```

响应的最大大小由 *subrequest\_output\_buffer\_size* 指令设置:

```
location /remote/ {  
    subrequest_output_buffer_size 64k;  
    # ...  
}
```

`set`

设置变量的值。命令具有以下参数:

`var`

变量名称。

`value`

变量值。如果赋值中包含变量, 则替换它们的值。

### 内置变量

`$date_local`

当前本地时区时间。格式由 `config` 命令和 `timefmt` 参数设置。

`$date_gmt`

当前 GMT 时间。格式由 `config` 命令和 `timefmt` 参数设置。

### SSL

提供必要的 HTTPS 支持。

当从 源代码构建时, 此模块默认未构建; 可以通过使用 `--with-http_ssl_module` 构建选项来启用它。

在来自 我们仓库的软件包和镜像中, 该模块已包含在构建中。

#### 重要

此模块需要 OpenSSL 库。

### 配置示例

为了减少处理器负载, 建议

- 将工作进程 的数量设置为与处理器数量相同,
- 启用保持活动 连接,
- 启用共享 会话缓存,

- 禁用内置 会话缓存,
- 并可能增加会话生命周期 (默认情况下为 5 分钟):

```
worker_processes auto;

http {

    # ...

    server {

        listen          443 ssl;
        keepalive_timeout 70;

        ssl_protocols   TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
        ssl_ciphers      AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate  /usr/local/angie/conf/cert.pem;
        ssl_certificate_key /usr/local/angie/conf/cert.key;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;

        # ...

    }
}
```

## 指令

### ssl\_buffer\_size

语法	<code>ssl_buffer_size size;</code>
默认值	<code>ssl_buffer_size 16k;</code>
上下文	http, server

设置用于发送数据的缓冲区大小。

默认情况下, 缓冲区大小为 16k, 这在发送大响应时对应于最小开销。为了最小化首字节时间, 可以使用较小的值, 例如:

```
ssl_buffer_size 4k;
```

## ssl\_certificate

语法	<code>ssl_certificate file [file];</code>
默认值	—
上下文	http, server

指定具有 PEM 格式证书的文件用于给定的虚拟服务器。如果除了主证书还应指定中间证书, 则应按以下顺序在同一文件中指定: 首先是主证书, 然后是中间证书。可以将 PEM 格式的私钥放在同一文件中。

此指令可以多次指定以加载不同类型的证书, 例如 RSA 和 ECDSA:

```
server {  
    listen          443 ssl;  
    server_name     example.com;  
  
    ssl_certificate example.com.rsa.crt;  
    ssl_certificate_key example.com.rsa.key;  
  
    ssl_certificate example.com.ecdsa.crt;  
    ssl_certificate_key example.com.ecdsa.key;  
  
    # ...  
}
```

只有 OpenSSL 1.0.2 或更高版本支持为不同证书提供单独的证书链。对于旧版本, 只能使用一个证书链。

### 重要

使用 OpenSSL 1.0.2 或更高版本时, 可以在文件名中使用变量:

```
ssl_certificate     $ssl_server_name.crt;  
ssl_certificate_key $ssl_server_name.key;
```

注意, 使用变量意味着每次 SSL 握手都会加载一个证书, 这可能会对性能产生负面影响。

可以将值“data:\$variable”指定为 file, 它从变量中加载证书而不使用中间文件。注意, 不当使用此语法可能具有安全隐患, 例如将密钥数据写入错误日志。

### 重要

应该记住, 由于 HTTPS 协议限制, 为了最大兼容性, 虚拟服务器应监听在不同的 IP 地址。

Added in version 1.2.0: 如果 `ssl_nTLS` 已启用, 指令可以接受两个参数 (密钥的签名和加密部分) 而不是一个:



```
listen ... ssl;

ssl_ntls on;

# 双NTLS证书
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规RSA证书一起使用
ssl_certificate      rsa.crt;
ssl_certificate_key  rsa.key;
```

### ssl\_certificate\_key

语法	ssl_certificate_key <i>file</i> [ <i>file</i> ];
默认值	—
上下文	http, server

指定具有 PEM 格式私钥的文件用于给定的虚拟服务器。

#### 重要

使用 OpenSSL 1.0.2 或更高版本时, 可以在文件名中使用变量。

可以将值“engine:name:id”指定为 `file`, 它从 OpenSSL 引擎名中加载具有指定 `id` 的私钥。

可以将值“data:\$variable”指定为 `file`, 它从变量中加载私钥而不使用中间文件。注意, 不当使用此语法可能具有安全隐患, 例如将密钥数据写入错误日志。

Added in version 1.2.0: 如果 `ssl_ntls` 已启用, 指令可以接受两个参数 (密钥的签名和加密部分) 而不是一个:

```
listen ... ssl;

ssl_ntls on;

# 双NTLS证书
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规RSA证书一起使用
ssl_certificate      rsa.crt;
ssl_certificate_key  rsa.key;
```

## ssl\_ciphers

语法	<code>ssl_ciphers <i>ciphers</i>;</code>
默认值	<code>ssl_ciphers HIGH:!aNULL:!MD5;</code>
上下文	http, server

指定启用的密码。密码格式为 OpenSSL 库理解的格式, 例如:

```
ssl_ciphers ALL:!aNULL:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

可以使用“`openssl ciphers`”命令查看完整列表。

## ssl\_client\_certificate

语法	<code>ssl_client_certificate <i>file</i>;</code>
默认值	—
上下文	http, server

指定具有受信任 CA 证书的 PEM 格式文件, 用于验证 客户端证书和 OCSP 响应 (如果启用了 `ssl_stapling`)。

证书列表将发送给客户端。如果不希望如此, 可以使用 `ssl_trusted_certificate` 指令。

## ssl\_conf\_command

语法	<code>ssl_conf_command <i>name value</i>;</code>
默认值	—
上下文	http, server

设置任意 OpenSSL 配置命令。

### 重要

此指令在使用 OpenSSL 1.0.2 或更高版本时受支持。

可以在同一级别指定多个 `ssl_conf_command` 指令:

```
ssl_conf_command Options PrioritizeChaCha;  
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

这些指令仅在当前级别未定义 `ssl_conf_command` 指令时, 从上一级配置继承。

### ⚠ 小心

直接配置 OpenSSL 可能导致意外行为。

## ssl\_crl

语法	<code>ssl_crl file;</code>
默认值	—
上下文	http, server

指定具有吊销证书 (CRL) 的 PEM 格式文件, 用于验证 客户端证书。

## ssl\_dhparam

语法	<code>ssl_dhparam file;</code>
默认值	—
上下文	http, server

指定具有 DHE 密码的 DH 参数的文件。

默认情况下未设置任何参数, 因此不会使用 DHE 密码。

## ssl\_early\_data

语法	<code>ssl_early_data on   off;</code>
默认值	<code>ssl_early_data off;</code>
上下文	http, server

启用或禁用 TLS 1.3 早期数据。

在早期数据中发送的请求可能面临 重放攻击。为防止此类攻击, 应用层应使用 `$ssl_early_data` 变量。

```
proxy_set_header Early-Data $ssl_early_data;
```

### 📌 重要

此指令在使用 OpenSSL 1.1.1 或更高版本以及 BoringSSL 时受支持。

## ssl\_ecdh\_curve

语法	<code>ssl_ecdh_curve curve;</code>
默认值	<code>ssl_ecdh_curve auto;</code>
上下文	<code>http, server</code>

指定用于 ECDHE 密码的曲线。

### 重要

当使用 OpenSSL 1.0.2 或更高版本时, 可以指定多个曲线, 例如:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

特殊值 `auto` 指示 Angie 在使用 OpenSSL 1.0.2 或更高版本时使用内置于 OpenSSL 库中的列表, 或在旧版本中使用 `prime256v1`。

### 重要

当使用 OpenSSL 1.0.2 或更高版本时, 此指令设置服务器支持的曲线列表。因此, 为了使 ECDSA 证书正常工作, 重要的是包括证书中使用的曲线。

## ssl\_ntls

Added in version 1.2.0.

语法	<code>ssl_ntls on   off;</code>
默认值	<code>ssl_ntls off;</code>
上下文	<code>http, server</code>

启用服务器端对使用 TongSuo 库的 NTLS 的支持。

```
listen ... ssl;  
ssl_ntls on;
```

### 重要

使用 `--with-ntls` 构建选项构建 Angie, 并链接到启用 NTLS 的 SSL 库

```
./configure --with-openssl=../Tongsuo-8.3.0 \  
            --with-openssl-opt=enable-ntls \  
            --with-ntls
```

## ssl\_ocsp

语法	<code>ssl_ocsp on   off   leaf;</code>
默认值	<code>ssl_ocsp off;</code>
上下文	http, server

启用客户端证书链的 OCSP 验证。参数 `leaf` 仅启用客户端证书的验证。

为了使 OCSP 验证正常工作, 必须将 `ssl_verify_client` 指令设置为 `on` 或 `optional`。

要解析 OCSP 响应者的主机名, 还应该指定 `resolver` 指令。

示例:

```
ssl_verify_client on;  
ssl_ocsp          on;  
resolver          127.0.0.53;
```

## ssl\_ocsp\_cache

语法	<code>ssl_ocsp_cache off   [shared:name:size];</code>
默认值	<code>ssl_ocsp_cache off;</code>
上下文	http, server

设置用于 OCSP 验证的客户端证书状态缓存的名称和大小。缓存在所有工作进程之间共享。可以在多个虚拟服务器中使用相同名称的缓存。

参数 `off` 禁止使用缓存。

## ssl\_ocsp\_responder

语法	<code>ssl_ocsp_responder uri;</code>
默认值	—
上下文	http, server

覆盖在 `validation` 客户端证书的 "Authority Information Access" 证书扩展中指定的 OCSP 响应者的 URL。

仅支持 `http://` OCSP 响应者:

```
ssl_ocsp_responder http://ocsp.example.com/;
```

### ssl\_password\_file

语法	<code>ssl_password_file file;</code>
默认值	—
上下文	http, server

指定包含 *secret keys* 的密码短语的文件, 其中每个密码短语单独占一行。在加载密钥时依次尝试这些密码短语。

示例:

```
http {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        server_name www1.example.com;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        server_name www2.example.com;

        # 可以使用命名管道代替文件
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

### ssl\_prefer\_server\_ciphers

语法	<code>ssl_prefer_server_ciphers on   off;</code>
默认值	<code>ssl_prefer_server_ciphers off;</code>
上下文	http, server

指定在使用 SSLv3 和 TLS 协议时应优先选择服务器密码而不是客户端密码。

## ssl\_protocols

语法	<code>ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];</code>
默认值	<code>ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;</code>
上下文	<code>http, server</code>

在 1.2.0 版本发生变更: TLSv1.3 参数添加到默认集合中。

启用指定的协议。

### 重要

仅当使用 OpenSSL 1.0.1 或更高版本时, TLSv1.1 和 TLSv1.2 参数才有效。

仅当使用 OpenSSL 1.1.1 或更高版本时, TLSv1.3 参数才有效。

## ssl\_reject\_handshake

语法	<code>ssl_reject_handshake on   off;</code>
默认值	<code>ssl_reject_handshake off;</code>
上下文	<code>http, server</code>

如果启用, 将拒绝 *server* 块中的 SSL 握手。

例如, 在以下配置中, 将拒绝与服务器名称不是 *example.com* 的 SSL 握手:

```
server {
    listen          443 ssl default_server;
    ssl_reject_handshake on;
}

server {
    listen          443 ssl;
    server_name     example.com;
    ssl_certificate example.com.crt;
    ssl_certificate_key example.com.key;
}
```

## ssl\_session\_cache

语法	<code>ssl_session_cache off   none   [ builtin[:size]] [ shared:name:size];</code>
默认值	<code>ssl_session_cache none;</code>
上下文	http, server

设置存储会话参数的缓存类型和大小。缓存可以是以下类型之一：

off	严格禁止使用会话缓存：Angie 明确告知客户端不允许重用会话。
none	温和禁止使用会话缓存：Angie 告知客户端可以重用会话，但实际上并不会在缓存中存储会话参数。
builtin	内建于 OpenSSL 的缓存；仅由一个工作进程使用。缓存大小以会话数指定。如果未给定大小，则默认为 20480 个会话。使用内建缓存可能导致内存碎片化。
shared	在所有工作进程之间共享的缓存。缓存大小以字节为单位指定；一兆字节可以存储大约 4000 个会话。每个共享缓存应有一个任意名称。可以在多个虚拟服务器中使用相同名称的缓存。还用于自动生成、存储和定期轮换 TLS 会话票据密钥，除非通过 <code>ssl_session_ticket_key</code> 指令明确配置。

可以同时使用两种缓存类型，例如：

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

但仅使用共享缓存而不使用内建缓存应该更有效。

## ssl\_session\_ticket\_key

语法	<code>ssl_session_ticket_key file;</code>
默认值	—
上下文	http, server

设置用于加密和解密 TLS 会话票据的密钥文件。如果需要在多个服务器之间共享相同的密钥，则此指令是必要的。默认情况下，使用随机生成的密钥。

如果指定了多个密钥，则仅使用第一个密钥来加密 TLS 会话票据。这允许配置密钥轮换，例如：

```
ssl_session_ticket_key current.key;  
ssl_session_ticket_key previous.key;
```

文件必须包含 80 或 48 字节的随机数据，可以使用以下命令创建：

```
openssl rand 80 > ticket.key
```

根据文件大小，使用 AES256（对于 80 字节密钥）或 AES128（对于 48 字节密钥）进行加密。



### ssl\_session\_tickets

语法	<code>ssl_session_tickets on   off;</code>
默认值	<code>ssl_session_tickets on;</code>
上下文	<code>http, server</code>

启用或禁用通过 TLS 会话票据 恢复会话。

### ssl\_session\_timeout

语法	<code>ssl_session_timeout <i>time</i>;</code>
默认值	<code>ssl_session_timeout 5m;</code>
上下文	<code>http, server</code>

指定客户端可以重用会话参数的时间。

### ssl\_stapling

语法	<code>ssl_stapling on   off;</code>
默认值	<code>ssl_stapling off;</code>
上下文	<code>http, server</code>

启用或禁用服务器对 OCSP 响应的捆绑。示例：

```
ssl_stapling on;  
resolver 127.0.0.53;
```

为了使 OCSP 捆绑正常工作，服务器证书颁发者的证书应该是已知的。如果 `ssl_certificate` 文件不包含中级证书，则服务器证书颁发者的证书应该存在于 `ssl_trusted_certificate` 文件中。

#### 注意

为了解 OCSP 响应者的主机名，还应该指定 `resolver` 指令。

### ssl\_stapling\_file

语法	<code>ssl_stapling_file file;</code>
默认值	—
上下文	http, server

设置后, 捆绑的 OCSP 响应将从指定文件中获取, 而不是查询服务器证书中指定的 OCSP 响应者。  
文件应为“`openssl ocsf`”命令生成的 DER 格式。

### ssl\_stapling\_responder

语法	<code>ssl_stapling_responder uri;</code>
默认值	—
上下文	http, server

覆盖 “Authority Information Access” 证书扩展中指定的 OCSP 响应者的 URL。

仅支持“`http://`” OCSP 响应者:

```
ssl_stapling_responder http://ocsp.example.com/;
```

### ssl\_stapling\_verify

语法	<code>ssl_stapling_verify on   off;</code>
默认值	<code>ssl_stapling_verify off;</code>
上下文	http, server

启用或禁用服务器对 OCSP 响应的验证。

为了使验证正常工作, 服务器证书颁发者的证书、根证书和所有中级证书都应该配置为可信的, 使用 `ssl_trusted_certificate` 指令。

### ssl\_trusted\_certificate

语法	<code>ssl_trusted_certificate file;</code>
默认值	—
上下文	http, server

指定一个包含以 PEM 格式的可信 CA 证书的文件, 用于 *verify* 客户端证书和 OCSP 响应, 如果启用了 *ssl\_stapling*。

与通过 *ssl\_client\_certificate* 设置的证书不同, 这些证书列表不会发送给客户端。

### ssl\_verify\_client

语法	<code>ssl_verify_client on   off   optional   optional_no_ca;</code>
默认值	<code>ssl_verify_client off;</code>
上下文	http, server

启用客户端证书的验证。验证结果存储在 *\$ssl\_client\_verify* 变量中。

<code>optional</code>	请求客户端证书并在证书存在时进行验证。
<code>optional_no_ca</code>	请求客户端证书, 但不要求其由受信任的 CA 证书签署。这适用于由 Angie 外部的服务执行实际证书验证的情况。

### ssl\_verify\_depth

语法	<code>ssl_verify_depth number;</code>
默认值	<code>ssl_verify_depth 1;</code>
上下文	http, server

设置客户端证书链的验证深度。

### 错误处理

`http_ssl` 模块支持若干非标准错误代码, 可用于使用 *error\_page* 指令进行重定向:

495	客户端证书验证期间发生错误;
496	客户端未提供所需的证书;
497	常规请求已发送到 HTTPS 端口。

重定向在请求完全解析后发生, 变量如 *\$request\_uri*, *\$uri*, *\$args* 等可用时进行。

指定要更改的代理服务器响应头中“Set-Cookie”字段的路径属性中的文本。重定向在请求完全解析后进行, 并且变量如 *\$request\_uri*, *\$uri*, *\$args* 和其他变量可用时进行。

## 内建变量

`http_ssl` 模块支持内建变量:

`$ssl_alpn_protocol`

返回 SSL 握手期间 ALPN 选择的协议, 否则返回空字符串。

`$ssl_cipher`

返回已建立 SSL 连接所使用的密码套件名称。

`$ssl_ciphers`

返回客户端支持的密码套件列表。已知密码套件按名称列出, 未知的以十六进制显示, 例如:

```
AES128-SHA:AES256-SHA:0x00ff
```

### 重要

该变量仅在使用 OpenSSL 版本 1.0.2 或更高版本时完全支持。对于旧版本, 该变量仅对新会话可用, 并且仅列出已知密码套件。

`$ssl_client_escaped_cert`

返回已建立 SSL 连接的客户端证书的 PEM 格式 (URL 编码);

`$ssl_client_fingerprint`

返回已建立 SSL 连接的客户端证书的 SHA1 指纹;

`$ssl_client_i_dn`

根据 RFC 2253, 返回已建立 SSL 连接的客户端证书的“颁发者 DN”字符串;

```
$ssl_client_i_dn_legacy
```

返回已建立 SSL 连接的客户端证书的“颁发者 DN”字符串。

```
$ssl_client_raw_cert
```

返回已建立 SSL 连接的客户端证书的 PEM 格式。

```
$ssl_client_s_dn
```

根据 RFC 2253, 返回已建立 SSL 连接的客户端证书的“主体 DN”字符串。

```
$ssl_client_s_dn_legacy
```

返回已建立 SSL 连接的客户端证书的“主体 DN”字符串。

```
$ssl_client_serial
```

返回已建立 SSL 连接的客户端证书的序列号。

```
$ssl_client_v_end
```

返回客户端证书的结束日期。

```
$ssl_client_v_remain
```

返回距离客户端证书过期的天数。

```
$ssl_client_v_start
```

返回客户端证书的开始日期。

```
$ssl_client_verify
```

返回客户端证书验证的结果: SUCCESS, "FAILED:reason", 如果没有证书则返回 NONE。

### `$ssl_curve`

返回用于 SSL 握手密钥交换过程的协商曲线。已知曲线按名称列出, 未知的以十六进制显示, 例如:

```
prime256v1
```

#### 重要

该变量仅在使用 OpenSSL 版本 3.0 或更高版本时支持。对于旧版本, 变量值将是空字符串。

### `$ssl_curves`

返回客户端支持的曲线列表。已知曲线按名称列出, 未知的以十六进制显示, 例如:

```
0x001d:prime256v1:secp521r1:secp384r1
```

#### 重要

该变量仅在使用 OpenSSL 版本 1.0.2 或更高版本时支持。对于旧版本, 变量值将是空字符串。

该变量仅对新会话可用。

### `$ssl_early_data`

如果使用 TLS 1.3 *early* 数据且握手未完成, 返回“1”, 否则返回“”。

### `$ssl_protocol`

返回已建立 SSL 连接的协议。

### `$ssl_server_cert_type`

根据服务器证书和密钥的类型, 取值为 RSA, DSA, ECDSA, ED448, ED25519, SM2, RSA-PSS, 或 unknown。

### `$ssl_server_name`

返回通过 SNI 请求的服务器名称;

`$ssl_session_id`

返回已建立 SSL 连接的会话标识符。

`$ssl_session_reused`

如果重复使用 SSL 会话, 返回 `r`, 否则返回 “.”。

## 简化状态

该模块提供对基本状态信息的访问。

当从源代码中构建时, 默认不会构建此模块; 应通过 `--with-http_stub_status_module` 构建选项来启用。

在来自 我们的仓库的软件包和镜像中, 该模块已包含在构建中。

## 配置示例

```
location = /basic_status {  
    stub_status;  
}
```

此配置创建了一个简单的网页, 显示基本状态数据, 如下所示:

```
活跃连接数: 291  
服务器接受 处理 请求  
16630948 16630948 31070465  
读取: 6 写入: 179 等待: 106
```

## 指令

### `stub_status`

语法	<code>stub_status;</code>
默认值	—
上下文	<code>server, location</code>

基本状态信息将可从所在位置访问。

## 数据

提供以下状态信息:

活跃连接数

当前活跃客户端连接数, 包括等待连接。

接受

接受的客户端连接总数。

处理

处理的连接总数。通常, 该参数值与接受数相同, 除非达到某些资源限制 (例如, *worker\_connections* 限制)。

请求

客户端请求总数。

读取

当前读取请求头的连接数。

写入

当前向客户端写回应的连接数。

等待

当前空闲等待请求的客户端连接数。



## 内置变量

`$connections_active`

与 活跃连接数的值相同;

`$connections_reading`

与 读取的值相同;

`$connections_writing`

与 写入的值相同;

`$connections_waiting`

与 等待的值相同。

替换 ###

该模块是一个过滤器, 通过将一个指定字符串替换为另一个字符串来修改响应。

当从源代码 构建时, 此模块默认不被构建; 它应通过 `--with-http_sub_module` 构建选项来启用。

在来自 我们的存储库的软件包和镜像中, 该模块已包含在构建中。

## 配置示例

```
location / {
    sub_filter '<a href="http://127.0.0.1:8080/' '<a href="https://$host/';
    sub_filter 'sub_filter <i>string replacement</i></code> ; |
| 默认值 | —                                                   |
| 上下文 | http, server, location                              |

设置要替换的字符串和替换字符串。匹配要替换的字符串时不区分大小写。要替换的字符串和替换字符串可以包含变量。在同一配置级别上可以指定多个 `sub_filter` 指令。只有在当前级别未定义 `sub_filter` 指令的情况下, 这些指令才会从上一级配置继承。

### sub\_filter\_last\_modified

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>sub_filter_last_modified on   off;</code> |
| 默认值 | <code>sub_filter_last_modified off;</code>      |
| 上下文 | http, server, location                          |

允许在替换期间保留原始响应的“Last-Modified”头字段, 以便于响应缓存。

默认情况下, 响应内容在处理过程中被修改, 头字段被移除。

### sub\_filter\_once

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>sub_filter_once on   off;</code> |
| 默认值 | <code>sub_filter_once on;</code>       |
| 上下文 | http, server, location                 |

指示是否仅查找每个字符串一次或多次。

### sub\_filter\_types

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>sub_filter_types mime-type ...;</code> |
| 默认值 | <code>sub_filter_types text/html;</code>     |
| 上下文 | http, server, location                       |

在指定的 MIME 类型的响应中启用字符串替换, 除了“`text/html`”。特殊值“\*”匹配任何 MIME 类型。

## 上游

该模块用于定义可以通过 `proxy_pass`、`fastcgi_pass`、`uwsgi_pass`、`scgi_pass`、`memcached_pass` 和 `grpc_pass` 指令引用的服务器组。

## 配置示例

```
upstream backend {
    zone backend 1m;
    server backend1.example.com      weight=5;
    server backend2.example.com:8080;
    server backend3.example.com      service=_example._tcp resolve;
    server unix:/tmp/backend3;

    server backup1.example.com:8080  backup;
    server backup2.example.com:8080  backup;
}

resolver 127.0.0.53 status_zone=resolver;

server {
    location / {
        proxy_pass http://backend;
    }
}
```

## 指令

### bind\_conn (PRO)

|     |                               |
|-----|-------------------------------|
| 语法  | <code>bind_conn value;</code> |
| 默认值 | —                             |
| 上下文 | upstream                      |

当 *value* 被设置为变量字符串, 且值不是 "" 和 "0" 时, 启用将服务器连接绑定到客户端。

#### ⚠ 注意

`bind_conn` 指令必须在设置负载均衡方法的所有指令之后使用; 否则将不起作用。如果同时使用了 *sticky*, `bind_conn` 应该出现在 *sticky* 之后。

#### ⚠ 注意

使用该指令时, 配置 *http\_proxy* 模块允许保持连接, 例如:

```
proxy_http_version 1.1;
proxy_set_header Connection "";
```

该指令的典型用例是代理 NTLM 认证连接, 当协商开始时, 客户端应与服务器绑定:

```
map $http_authorization $ntlm {
    ~*^N(?:TLM|egotiate) 1;
}

upstream ntlm_backend {
    server 127.0.0.1:8080;
    bind_conn $ntlm;
}

server {
    # ...
    location / {
        proxy_pass http://ntlm_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        # ...
    }
}
```

## feedback (PRO)

Added in version 1.6.0: PRO

|     |                                                                                                                                                  |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>feedback</code> <i>variable</i> [ <i>inverse</i> ] [ <i>factor=number</i> ] [ <i>account=condition_variable</i> ]<br>[ <i>last_byte</i> ]; |
| 默认值 | —                                                                                                                                                |
| 上下文 | upstream                                                                                                                                         |

为 `upstream` 启用基于反馈的负载均衡机制。它动态调整负载均衡决策, 通过将每个对等体的权重乘以其平均反馈值来实现, 反馈值受到 *variable* 的时间值的影响, 并受可选条件的约束。

接受以下参数:

|          |                                                                                                                                            |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------|
| variable | 从中获取反馈值的变量。它应代表性能或健康指标，并且应由对等体通过头字段或其他方式提供。<br>该值在每次来自对等体的响应时被评估，并根据 <code>inverse</code> 和 <code>factor</code> 设置融入滚动平均值。                 |
| inverse  | 如果设置，反馈值将被解释为相反的，即较低的值表示更好的性能。                                                                                                             |
| factor   | 反馈值在计算平均值时的权重因子。有效值为 0 到 99 之间的整数。默认值为 90。<br>平均反馈是使用 <a href="#">指数移动平均</a> 公式计算的。<br>因子越大，平均值受新值影响越小；如果因子设置为 90，结果包含 90% 的先前值和仅 10% 的新值。 |
| account  | 指定一个条件变量，控制哪些响应应包含在计算中。平均值仅在响应的条件变量不为 "" 或 "0" 时更新。                                                                                        |

**备注**

默认情况下，来自 `probes` 的响应不包含在计算中；将 `$upstream_probe` 变量与 `account` 相结合可以包括这些响应，甚至排除其他所有响应。

|           |                                    |
|-----------|------------------------------------|
| last_byte | 允许在接收到完整响应后处理来自上游服务器的反馈，而不仅仅是在头之后。 |
|-----------|------------------------------------|

示例：

```
upstream backend {  
  
    zone backend 1m;  
  
    feedback $feedback_value factor=80 account=$condition_value;  
  
    server backend1.example.com;  
    server backend2.example.com;  
}  
  
map $upstream_http_custom_score $feedback_value {  
    "high"           100;  
    "medium"        75;  
    "low"           50;  
    default          10;  
}  
  
map $upstream_probe $condition_value {  
    "high_priority" "1";  
    "low_priority"  "0";  
    default         "1";  
}
```

这将根据从响应头字段获得的特定分数将服务器响应分类到不同的反馈级别，并且还添加了从 `$upstream_probe` 映射的条件，仅考虑来自 `high_priority` 探针的响应或常规客户端请求的响应。

## hash

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>hash key [consistent];</code> |
| 默认值 | —                                   |
| 上下文 | upstream                            |

指定服务器组的负载均衡方法, 其中客户端与服务器的映射基于哈希键值。键可以包含文本、变量及其组合。请注意, 添加或移除服务器可能导致大部分键重新映射到不同的服务器。该方法与 `Cache::Memcached` Perl 库兼容。

如果指定了 `consistent` 参数, 将使用 `ketama` 一致性哈希方法。该方法确保在添加或移除服务器时, 仅有少量键会重新映射到不同的服务器。这有助于提高缓存服务器的缓存命中率。该方法与 `Cache::Memcached::Fast` Perl 库兼容, 并设置 `ketama_points` 参数为 160。

## ip\_hash

|     |                       |
|-----|-----------------------|
| 语法  | <code>ip_hash;</code> |
| 默认值 | —                     |
| 上下文 | upstream              |

指定一个组应使用基于客户端 IP 地址的负载均衡方法。使用客户端 IPv4 地址的前三个八位字节或整个 IPv6 地址作为哈希键。该方法确保来自同一客户端的请求将始终传递到同一服务器, 除非该服务器不可用。在这种情况下, 客户端请求将传递到另一台服务器。大多数情况下, 它将始终是同一台服务器。

如果需要暂时移除其中一台服务器, 应使用 `down` 参数标记, 以保留当前的客户端 IP 地址哈希。

```
upstream backend {
    ip_hash;

    server backend1.example.com;
    server backend2.example.com;
    server backend3.example.com down;
    server backend4.example.com;
}
```

## keepalive

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>keepalive connections;</code> |
| 默认值 | —                                   |
| 上下文 | upstream                            |

启用到上游服务器连接的缓存。

`connections` 参数设置每个工作进程中空闲保持连接到上游服务器的最大数量。当超过此数量时，最不常用的连接将被关闭。

### 备注

特别需要注意的是, `keepalive` 指令并不限制工作进程可以打开的上游服务器连接总数。`connections` 参数应设置为足够小的值, 以便上游服务器处理新的传入连接。

### 注意

`keepalive` 指令必须在设置负载均衡方法的所有指令之后使用; 否则将不起作用。

使用保持连接的 memcached 上游配置示例:

```
upstream memcached_backend {
    server 127.0.0.1:11211;
    server 10.0.0.2:11211;

    keepalive 32;
}

server {
    #...

    location /memcached/ {
        set $memcached_key $uri;
        memcached_pass memcached_backend;
    }
}
```

对于 HTTP, 应将 `proxy_http_version` 指令设置为“1.1”, 并清除“Connection” 头字段:

```
upstream http_backend {
    server 127.0.0.1:8080;
```

```
    keepalive 16;
}

server {
    #...

    location /http/ {
        proxy_pass http://http_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        # ...
    }
}
```

#### **i** 备注

或者, 可以通过向上游服务器传递"Connection: Keep-Alive" 头字段来使用 HTTP/1.0 持久连接, 尽管这种方法不推荐。

对于 FastCGI 服务器, 需要设置`fastcgi_keep_conn` 以使保持连接生效:

```
upstream fastcgi_backend {
    server 127.0.0.1:9000;

    keepalive 8;
}

server {
    #...

    location /fastcgi/ {
        fastcgi_pass fastcgi_backend;
        fastcgi_keep_conn on;
        # ...
    }
}
```

#### **i** 备注

SCGI 和 uwsgi 协议没有保持连接的概念。



### keepalive\_requests

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>keepalive_requests number;</code> |
| 默认值 | <code>keepalive_requests 1000;</code>   |
| 上下文 | upstream                                |

设置可以通过一个保持连接服务的最大请求数。在达到最大请求数后, 连接将关闭。

定期关闭连接是为了释放每个连接的内存分配。因此, 使用过高的最大请求数可能导致过多的内存使用, 不推荐这样做。

### keepalive\_time

|     |                                   |
|-----|-----------------------------------|
| 语法  | <code>keepalive_time time;</code> |
| 默认值 | <code>keepalive_time 1h;</code>   |
| 上下文 | upstream                          |

限制通过一个保持连接处理请求的最长时间。在达到此时间后, 连接将在后续请求处理后关闭。

### keepalive\_timeout

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>keepalive_timeout 时间;</code>  |
| 默认值 | <code>keepalive_timeout 60s;</code> |
| 上下文 | upstream                            |

设置一个超时时间, 在此期间, 与上游服务器的空闲 keepalive 连接将保持打开状态。

### least\_conn

|     |                          |
|-----|--------------------------|
| 语法  | <code>least_conn;</code> |
| 默认值 | —                        |
| 上下文 | upstream                 |

指定一个组应该使用一种负载均衡方法, 将请求传递给活动连接最少的服务器, 同时考虑服务器的权重。如果有多个这样的服务器, 它们将按照加权轮询的方式依次尝试。

## least\_time (PRO)

|     |                                                                          |
|-----|--------------------------------------------------------------------------|
| 语法  | <code>least_time header   last_byte [factor= 数字] [account= 条件变量];</code> |
| 默认值 | —                                                                        |
| 上下文 | upstream                                                                 |

指定该组应使用一种负载均衡方法, 其中活动服务器接收请求的机会与其平均响应时间成反比; 响应时间越短, 服务器接收的请求就越多。

|           |                  |
|-----------|------------------|
| header    | 指令仅计算响应头。        |
| last_byte | 指令使用接收整个响应的平均时间。 |

Added in version 1.7.0: PRO

|         |                                                           |
|---------|-----------------------------------------------------------|
| factor  | 具有与 <i>response_time_factor (PRO)</i> 相同的目的, 并在设置时覆盖它。    |
| account | 指定一个条件变量, 以控制应包含在计算中的响应。只有当响应的条件变量不为 "" 或 "0" 时, 平均值才会更新。 |

**备注**

默认情况下, 来自 *probes* 的响应不会包含在计算中; 将 *\$upstream\_probe* 变量与 *account* 结合使用, 可以包含这些响应, 甚至排除其他所有内容。

相应的移动平均值, 经过 *factor* 和 *account* 的调整, 也在服务器的 *health* 对象中作为 *header\_time* 和 *response\_time* 呈现, 位于 API 中的 *upstream metrics* 中。

## queue (PRO)

Added in version 1.4.0: PRO

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>queue 数量 [timeout= 时间];</code> |
| 默认值 | —                                    |
| 上下文 | upstream                             |

如果无法在第一次尝试时将请求分配给代理服务器 (例如, 在短暂的服务中断期间或负载激增达到 *max\_conns* 限制时), 请求不会被拒绝; 相反, *Angie* 尝试将其排队等待处理。

指令中的数字设置了 *worker process* 的队列中请求的最大数量。如果队列已满, 将向客户端返回 502 (Bad Gateway) 错误。

**i** 备注

`proxy_next_upstream` 指令的逻辑也适用于排队请求。具体而言, 如果为请求选择了一个服务器, 但无法将其传递给该服务器, 请求可能会被返回到队列中。

如果在由 `timeout` 设置的 时间内未选择服务器处理排队请求 (默认为 60 秒), 将向客户端返回 502 (Bad Gateway) 错误。提前关闭连接的客户端请求也将从队列中移除; 队列中请求的计数器在 *API* 中提供。

**A** 注意

必须在所有设置负载均衡方法的指令之后使用 `queue` 指令, 否则将无效。

**random**

|     |                            |
|-----|----------------------------|
| 语法  | <code>random [two];</code> |
| 默认值 | —                          |
| 上下文 | upstream                   |

指定一个组应该使用一种负载均衡方法, 将请求传递给随机选择的服务器, 同时考虑服务器的权重。

可选的 `two` 参数指示 *Angie* 随机选择两个服务器, 然后使用指定的方法选择一个服务器。默认方法是 `least_conn`, 将请求传递给活动连接最少的服务器。

**response\_time\_factor (PRO)**

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>response_time_factor 数字;</code> |
| 默认值 | <code>response_time_factor 90;</code> |
| 上下文 | upstream                              |

如果使用了 `least_time (PRO)` 负载均衡方法, 则设置计算平均响应时间时对 前一个值的平滑因子, 使用指数移动平均 公式。

数字越大, 新值对平均值的影响就越小; 如果数字设置为 90, 结果有 90% 的前一个值和仅 10% 的新值。允许的范围是 0 到 99 (含)。

相应的移动平均值在服务器的 `health` 对象中作为 `header_time` (仅头) 和 `response_time` (整个响应) 呈现, 位于 *API* 中的 `upstream metrics` 中。

**i** 备注

计算仅考虑成功的响应; 什么被认为是不成功的响应由 `proxy_next_upstream`、`fastcgi_next_upstream`、`uwsgi_next_upstream`、`scgi_next_upstream`、`memcached_next_upstream` 和 `grpc_next_upstream` 指令定义。此外, 仅当所有头接收并处理时, `header_time` 才会更新, 并且仅当整个响应接收时, `response_time` 才会更新。

## server

|     |                              |
|-----|------------------------------|
| 语法  | <code>server</code> 地址 [参数]; |
| 默认值 | —                            |
| 上下文 | upstream                     |

定义服务器的地址和其他参数。地址可以指定为域名或 IP 地址, 带有可选端口, 或者作为在 `unix:` 前缀之后指定的 UNIX 域套接字路径。如果未指定端口, 则使用端口 80。解析为多个 IP 地址的域名一次定义多个服务器。

可以定义以下参数:

|                            |                                                                  |
|----------------------------|------------------------------------------------------------------|
| <code>weight=</code> 数字    | 设置服务器的权重默认为 1。                                                   |
| <code>max_conns=</code> 数字 | 限制到代理服务器的最大同时活动连接数。默认值为 0, 表示没有限制。如果服务器组不驻留在共享内存中, 则限制对每个工作进程生效。 |

### 备注

如果启用了空闲 `keepalive` 连接、多个 `worker` 和共享内存, 则到代理服务器的活动和空闲连接总数可能会超过 `max_conns` 值。

`max_fails=` 数字—设置在 `fail_timeout` 期间与服务器通信不成功的尝试次数, 以认为服务器不可用; 然后将在相同的持续时间后重试。

什么被认为是不成功的尝试由 `proxy_next_upstream`、`fastcgi_next_upstream`、`uwsgi_next_upstream`、`scgi_next_upstream`、`memcached_next_upstream`、和 `grpc_next_upstream` 指令定义。

当达到 `max_fails` 时, `upstream_probe (PRO)` 探测也会认为该对等体不健康; 除非探测认为其再次健康, 否则不会接收客户端请求。

### 备注

如果上游中的一个 `server` 解析为多个对等体, 其 `max_fails` 设置适用于每个对等体。

如果上游在解析其所有 `server` 指令后只包含一个对等体, 则 `max_fails` 设置无效并将被忽略。

|                          |            |
|--------------------------|------------|
| <code>max_fails=1</code> | 默认的不成功尝试次数 |
| <code>max_fails=0</code> | 禁用尝试的计数    |

`fail_timeout=` 时间—设置在此期间与服务器通信的多次不成功尝试 (`max_fails`) 应发生的时间段, 以认为服务器不可用。然后服务器在相同的时间段内不可用, 然后再重试。

默认情况下, 这设置为 10 秒。

**i 备注**

如果上游中的一个 `server` 解析为多个对等体, 其 `fail_timeout` 设置适用于每个对等体。  
 如果上游在解析其所有 `server` 指令后只包含一个对等体, 则 `fail_timeout` 设置无效并将被忽略。

|                     |                                                                               |
|---------------------|-------------------------------------------------------------------------------|
| <code>backup</code> | 将服务器标记为备份服务器。当主要服务器不可用时, 将传递请求给它。                                             |
| <code>down</code>   | 将服务器标记为永久不可用。                                                                 |
| <code>drain</code>  | 将服务器设置为排水; 这意味着它仅接收以前使用 <i>sticky</i> 绑定的会话的请求。否则, 其行为类似于 <code>down</code> 。 |

**⚠ 小心**

参数 `backup` 不能与 `hash`, `ip_hash` 和 `random` 负载均衡方法一起使用。  
`down` 和 `drain` 选项是互斥的。

Added in version 1.1.0.

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>resolve</code>     | 启用监控与域名对应的 IP 地址列表的变化, 无需重新加载配置即可更新。该组应存储在共享内存区域中; 您还需要定义一个 <i>resolver</i> 。                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>service= 名称</code> | <p>启用解析 DNS SRV 记录并设置服务名称。要使此参数生效, 请指定 <code>resolve</code> 服务器参数, 提供不带端口号的主机名。</p> <p>如果服务名称中没有点, 名称将按照 RFC 标准形成: 服务名称以 <code>_</code> 为前缀, 然后在点后添加 <code>_tcp</code>。因此, 服务名称 <code>http</code> 将导致 <code>_http._tcp</code>。</p> <p>Angie 通过组合规范化的服务名称和主机名来解析 SRV 记录, 并通过 DNS 获得该组合的服务器列表, 以及它们的优先级和权重。</p> <ul style="list-style-type: none"> <li>最高优先级的 SRV 记录 (共享最小优先级值的记录) 解析为主要服务器, 其他记录成为备份服务器。如果在 <code>server</code> 中设置了 <code>backup</code>, 最高优先级的 SRV 记录解析为备份服务器, 而其他记录被忽略。</li> </ul> |

- 权重影响服务器的选择, 根据分配的容量: 权重越高接收到的请求越多。如果由 `server` 指令和 SRV 记录同时设置, 使用 `server` 设置的权重。

此示例将查询 `_http._tcp.backend.example.com` 记录:

```
server backend.example.com service=http resolve;
```

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <code>sid=id</code> | 设置组内的服务器 ID。如果省略参数, ID 将被设置为 IP 地址和端口或 UNIX 域套接字路径的十六进制 MD5 哈希值。 |
|---------------------|------------------------------------------------------------------|

Added in version 1.4.0.

|                              |                                                                                                                                                                                                                                                                       |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>slow_start=time</code> | <p>设置服务器重新上线时恢复 <code>weight</code> 的时间, 如果负载均衡使用轮循或 <code>least_conn</code> 方法。</p> <p>如果设置了该值, 并且服务器再次被视为可用和健康(由 <code>max_fails &lt;max_fails&gt;</code> 和 <code>upstream_probe (PRO)</code> 定义), 服务器将在分配的时间内逐渐恢复其指定的权重。</p> <p>如果未设置该值, 在类似情况下服务器将立即恢复其指定的权重。</p> |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### **i** 备注

如果一个上游中只有一个 `server`, `slow_start` 将无效并被忽略。

### state (PRO)

Added in version 1.2.0: PRO

|     |                          |
|-----|--------------------------|
| 语法  | <code>state file;</code> |
| 默认值 | —                        |
| 上下文 | <code>upstream</code>    |

指定保存上游服务器列表的 `file`。在从我们的包安装时, 会创建一个指定的目录 `/var/lib/angie/state/` (FreeBSD 上为 `/var/db/angie/state/`), 并赋予适当的权限来存储这些文件, 因此您只需在配置中添加文件的基本名称:

```
upstream backend {
    zone backend 1m;
    state /var/lib/angie/state/<FILE NAME>;
}
```

此服务器列表的格式类似于 `server`。文件的内容在通过配置 API 对 `/config/http/upstreams/` 部分的服务器进行任何修改时都会改变。文件在 Angie 启动或配置重载时读取。

### 小心

为了在 `upstream` 块中使用 `state` 指令, 该块不应有 `server` 指令; 相反, 必须具有共享内存区域 (`zone`)。

## sticky

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

|     |                                                                                                                                                                            |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <pre>sticky cookie name [attr=value]...; sticky route \$variable...; sticky learn zone=zone create=\$create_var1... lookup=\$lookup_var1... [header] [timeout=time];</pre> |
| 默认值 | —                                                                                                                                                                          |
| 上下文 | upstream                                                                                                                                                                   |

配置客户端会话到代理服务器的绑定, 模式由第一个参数指定; 要从定义了 `sticky` 的服务器中排空请求, 请在 `server` 块中使用 `drain` 选项。

### 注意

`sticky` 指令必须在设置负载均衡方法的所有指令之后使用; 否则, 将不起作用。如果也使用 `bind_conn (PRO)`, 则 `bind_conn` 应在 `sticky` 之后出现。

## cookie 模式

此模式使用 `cookie` 维护会话持久性。更适合于已经使用 `cookie` 管理会话的情况。

在此情况下, 尚未绑定到任何服务器的客户端请求将根据配置的负载均衡方法发送到一个服务器。此外, Angie 会设置一个唯一值标识服务器的 `cookie`。

`cookie` 的名称 (`name`) 由 `sticky` 指令设置, 值 (`value`) 对应于 `server` 指令的 `sid` 参数。如果设置了 `sticky_secret` 指令, 参数将被另外哈希。

包含此 `cookie` 的后续客户端请求将转发到由 `cookie` 的值标识的服务器, 即具有指定 `sid` 的服务器。如果选择服务器失败或选定服务器无法处理请求, 将根据配置的负载均衡方法选择其他服务器。

该指令允许为 `cookie` 分配属性; 默认设置的唯一属性是 `path=/`。属性值指定为带变量的字符串。要删除属性, 请为其设置空值: `attr=`。因此, `sticky cookie path=` 创建一个没有 `path` 的 `cookie`。

在此情况下, Angie 创建一个名为 `srv_id` 的 cookie, 寿命为一小时, 并具有变量指定的域:

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;

    sticky cookie srv_id domain=$my_domain max-age=3600;
}
```

### route 模式

此模式使用预定义的路由标识符, 可以嵌入在 URL、cookie 或其他请求属性中。因依赖于预定义值而较不灵活, 但如果这些标识符已经存在可能更适合。

在此情况下, 当代理服务器收到请求时, 可以为客户端分配一个路由, 并以客户端和服务器都知道的方式返回其标识符。`server` 指令的 `sid` 参数值必须用作路由标识符。如果设置了 `sticky_secret` 指令, 参数将被另外哈希。

希望使用此路由的客户端的后续请求必须包含服务器发出的标识符, 以确保它进入 Angie 变量, 例如在 `cookie` 或请求参数中。

指令列出了用于路由的特定变量。为选择要转发传入请求的服务器, 使用第一个非空变量; 然后将其与 `server` 指令的 `sid` 参数进行比较。如果选择服务器失败或选定服务器无法处理请求, 将根据配置的负载均衡方法选择其他服务器。

在此情况下, Angie 在 `route cookie` 中查找路由标识符, 然后在 `route` 请求参数中查找:

```
upstream backend {
    server backend1.example.com:8080 "sid=server 1";
    server backend2.example.com:8080 "sid=server 2";

    sticky route $cookie_route $arg_route;
}
```

### learn 模式 (PRO 1.4.0+)

此模式使用动态生成的密钥将客户端与特定代理服务器相关联; 更灵活, 因为它可以动态分配服务器, 在共享内存区域存储会话, 并支持不同的会话标识符传递方式。

在此情况下, 会话基于代理服务器的响应创建。`create` 和 `lookup` 参数列出变量, 指示如何创建新会话以及如何查找现有会话。两个参数可以多次出现。

会话标识符是使用 `create` 指定的第一个非空变量的值; 例如, 这可以是来自代理服务器的 `cookie`。

会话存储在共享内存区域; 其名称和大小由 `zone` 参数设置。如果会话在 `timeout` 设置的时间内处于非活动状态, 则会被删除。默认是 10 分钟。

希望使用会话的客户端的后续请求必须包含其标识符, 以确保它进入指定为 `lookup` 的非空变量中; 其值将与共享内存中的会话匹配。如果选择服务器失败或选定服务器无法处理请求, 将根据配置的负载均衡方法选择其他服务器。



`header` 参数允许在接收来自代理服务器的头部后立即创建会话。如果不使用它, 会话仅在处理请求后创建。

在此示例中, Angie 创建了一个会话, 在响应中设置了一个名为 `examplecookie` 的 cookie:

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;

    sticky learn
        create=$upstream_cookie_examplecookie
        lookup=$cookie_examplecookie
        zone=client_sessions:1m;
}
```

### `sticky_secret`

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>sticky_secret string;</code> |
| 默认值 | —                                  |
| 上下文 | upstream                           |

在 `cookie` 和 `route` 模式下, 为 `sticky` 指令的 MD5 哈希函数添加 `string` 作为盐值。 `string` 可以包含变量, 例如 `$remote_addr`:

```
upstream backend {
    server backend1.example.com:8080;
    server backend2.example.com:8080;

    sticky cookie cookie_name;
    sticky_secret my_secret.$remote_addr;
}
```

盐附加到被哈希的值; 要独立验证哈希机制:

```
$ echo -n "<VALUE><SALT>" | md5sum
```

## sticky\_strict

Added in version 1.2.0: Angie

Added in version 1.1.0-P1: Angie PRO

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>sticky_strict on   off;</code> |
| 默认值 | <code>sticky_strict off;</code>      |
| 上下文 | <code>upstream</code>                |

启用时, 如果所需服务器不可用, Angie 将返回 HTTP 502 错误给客户端, 而不是使用上游中任何其他可用服务器。

## upstream

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>upstream name { ... }</code> |
| 默认值 | —                                  |
| 上下文 | <code>http</code>                  |

定义一组服务器。服务器可以监听不同的端口。此外, 监听 TCP 和 UNIX 域套接字的服务器可以混合。

示例:

```
upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080      max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;

    server backup1.example.com backup;
}
```

默认情况下, 请求使用加权轮询平衡方法在服务器之间分配。在上述示例中, 每 7 个请求将按以下方式分配: 5 个请求发送到 `backend1.example.com`, 第二和第三个服务器各接收到一个请求。

如果与服务器通信时发生错误, 请求将传递给下一个服务器, 依此类推, 直到尝试所有正常工作的服务器为止。如果无法从任何服务器获得成功响应, 客户端将收到与最后一个服务器通信的结果。

## zone

|        |                                |
|--------|--------------------------------|
| Syntax | <code>zone name [size];</code> |
| 默认     | —                              |
| 上下文    | upstream                       |

定义共享内存区域的名称和大小, 该区域保存工作进程之间共享的组的配置和运行时状态。多个组可以共享同一个区域。在这种情况下, 只需指定一次大小即可。

## 内置变量

`http_upstream` 模块支持以下内置变量:

### `$upstream_addr`

保存上游服务器的 IP 地址和端口, 或者 UNIX 域套接字的路径。如果在请求处理中联系了多个服务器, 它们的地址用逗号分隔, 例如:

```
192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock
```

如果从一个服务器组到另一个服务器组发生内部重定向, 由“X-Accel-Redirect”或`error_page`发起, 则来自不同组的服务器地址用冒号分隔, 例如:

```
192.168.1.1:80, 192.168.1.2:80, unix:/tmp/sock : 192.168.10.1:80, 192.168.10.2:80
```

如果无法选择服务器, 该变量将保存服务器组的 `name`。

### `$upstream_bytes_received`

从上游服务器接收到的字节数。多个连接的值用逗号和冒号分隔, 类似于`$upstream_addr`变量中的地址。

### `$upstream_bytes_sent`

发送到上游服务器的字节数。多个连接的值用逗号和冒号分隔, 类似于`$upstream_addr`变量中的地址。

### `$upstream_cache_status`

保存访问响应缓存的状态。状态可以是 MISS、BYPASS、EXPIRED、STALE、UPDATING、REVALIDATED 或 HIT:

- MISS: 响应未在缓存中找到, 请求转发到上游服务器。
- BYPASS: 缓存被绕过, 请求直接转发到上游服务器。

- EXPIRED: 缓存的响应已过期, 并向上游服务器发送更新内容的新请求。
- STALE: 缓存的响应已过期, 但仍会提供给客户端, 直到从上游服务器最终获取更新。
- UPDATING: 缓存的响应已过期, 但仍会提供给客户端, 直到从上游服务器进行的当前更新完成。
- REVALIDATED: 缓存的响应已过期, 但已成功重新验证, 无需从上游服务器更新。
- HIT: 响应已从缓存提供。

如果缓存完全被绕过而没有访问, 则变量未设置。

#### `$upstream_connect_time`

保存与上游服务器建立连接所花费的时间; 时间以秒为单位, 精确到毫秒。在 SSL 的情况下, 包括握手时间。多个连接的时间用逗号和冒号分隔, 类似于 `$upstream_addr` 变量中的地址。

#### `$upstream_cookie_<name>`

在“Set-Cookie”响应头字段中由上游服务器发送的具有指定 `name` 的 cookie。仅保存最后一个服务器响应的 cookie。

#### `$upstream_header_time`

保存从上游服务器接收响应头所花费的时间; 时间以秒为单位, 精确到毫秒。多个响应的时间用逗号和冒号分隔, 类似于 `$upstream_addr` 变量中的地址。

#### `$upstream_http_<name>`

存储服务器响应头字段。例如, “Server”响应头字段可以通过 `$upstream_http_server` 变量获得。将头字段名称转换为变量名称的规则与以“\$http\_”前缀开头的变量相同。仅保存来自最后一个服务器响应的头字段。

#### `$upstream_queue_time`

保存请求在选择服务器之前在队列中花费的时间; 时间以秒为单位, 精确到毫秒。多个选择尝试的时间用逗号和冒号分隔, 类似于 `$upstream_addr` 变量中的地址。

### `$upstream_response_length`

保存从上游服务器获取的响应长度；长度以字节为单位。多个响应的长度用逗号和冒号分隔，类似于 `$upstream_addr` 变量中的地址。

### `$upstream_response_time`

保存从上游服务器接收响应所花费的时间；时间以秒为单位，精确到毫秒。多个响应的时间用逗号和冒号分隔，类似于 `$upstream_addr` 变量中的地址。

### `$upstream_status`

保存从上游服务器获取的响应的状态码。多个响应的状态码用逗号和冒号分隔，类似于 `$upstream_addr` 变量中的地址。如果无法选择服务器，该变量将保存 502（错误网关）状态码。

### `$upstream_sticky_status`

粘性请求的状态。

|      |                           |
|------|---------------------------|
| ""   | 请求发送到上游，没有启用粘性。           |
| NEW  | 没有粘性信息的请求。                |
| HIT  | 带有粘性信息的请求路由到所需的后端。        |
| MISS | 带有粘性信息的请求路由到由负载均衡算法选择的后端。 |

来自多个连接的值用逗号和冒号分隔，类似于 `$upstream_addr` 变量中的地址。

### `$upstream_trailer_<name>`

存储从上游服务器获取的响应末尾的字段。

## 上游探针

该模块实现了针对 `http_upstream` 的主动健康探测。

## 配置示例

```
server {
    listen ...;

    location @probes {
        ...
        proxy_pass http://backend;

        upstream_probe backend_probe
            uri=/probe
            port=10004
            interval=5s
            test=$good
            essential
            fails=3
            passes=3
            max_body=10m
            mode=idle;
    }
}
```

## 指令

### upstream\_probe (PRO)

Added in version 1.2.0: PRO

|     |                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>upstream_probe</code> <i>name</i> [ <code>uri=address</code> ] [ <code>port=number</code> ] [ <code>interval=time</code> ]<br>[ <code>method=method</code> ] [ <code>test=condition</code> ] [ <code>essential</code> [ <code>persistent</code> ]] [ <code>fails=number</code> ]<br>[ <code>passes=number</code> ] [ <code>max_body=number</code> ] [ <code>mode=always   idle   onfail</code> ]; |
| 默认  | —                                                                                                                                                                                                                                                                                                                                                                                                       |
| 上下文 | location                                                                                                                                                                                                                                                                                                                                                                                                |

为在相同 location 上下文中指定的 `proxy_pass`、`uwsgi_pass` 等的 `upstream` 组内的节点定义一个主动健康探测。接下来, Angie 会定期根据此处配置的参数探测上游组的每个节点。

如果请求成功并考虑了 `upstream_probe` 指令的所有参数设置, 以及指令所在 location 上下文对上游使用的控制设置 (如 `proxy_next_upstream` 和 `uwsgi_next_upstream` 指令等, 以及 `proxy_set_header` 等), 则节点的探测通过。

要使用探测, 上游必须有一个共享内存区 (`zone`)。一个上游可以配置多个探测。

接受以下参数:

|            |                                                                                                                                                                                                                                                           |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name       | 探测的必需名称。                                                                                                                                                                                                                                                  |
| uri        | 要添加到 <code>proxy_pass</code> 、 <code>uwsgi_pass</code> 等参数中的请求 URI。默认值 <code>-/</code> 。                                                                                                                                                                  |
| port       | 探测请求的替代端口号。                                                                                                                                                                                                                                               |
| interval   | 探测之间的间隔。默认值 <code>5s</code> 。                                                                                                                                                                                                                             |
| method     | 探测的 HTTP 方法。默认值 <code>GET</code> 。                                                                                                                                                                                                                        |
| test       | 探测的条件, 定义为一个包含变量的字符串。如果变量替换结果为 <code>"</code> 或 <code>0</code> , 则探测不通过。                                                                                                                                                                                  |
| essential  | 如果设置, 节点的初始状态正在被检查, 因此在探测通过前节点不会接收客户端请求。                                                                                                                                                                                                                  |
| persistent | 此参数需要首先启用 <code>essential</code> ; 在配置重载前被认为健康的 <code>persistent</code> 节点开始接收请求, 而无需首先通过此探测。                                                                                                                                                             |
| fails      | 将节点标记为不健康的连续失败探测次数。默认值 <code>1</code> 。                                                                                                                                                                                                                   |
| passes     | 将节点标记为健康的连续通过探测次数。默认值 <code>1</code> 。                                                                                                                                                                                                                    |
| max_body   | 响应体的最大内存量。默认值 <code>256k</code> 。                                                                                                                                                                                                                         |
| mode       | 探测模式, 取决于节点的健康状况: <ul style="list-style-type: none"><li><code>always</code> — 无论节点状态如何都进行探测;</li><li><code>idle</code> — 探测不健康的节点以及自上次客户端请求以来的 <code>interval</code> 已过期的节点。</li><li><code>onfail</code> — 仅探测不健康的节点。</li></ul> 默认值 <code>always</code> 。 |

示例:

```
upstream backend {
    zone backend 1m;

    server backend1.example.com;
    server backend2.example.com;
}

map $upstream_status $good {
    200    "1";
}

server {
    listen ...;

    location @probes {
        ...
        proxy_pass http://backend;

        upstream_probe backend_probe
            uri=/probe
            port=10004
            interval=5s
            test=$good
    }
}
```

```
    essential
    persistent
    fails=3
    passes=3
    max_body=10m
    mode=idle;
}
}
```

探测操作的详细信息:

- 最初, 在通过为其配置的所有 `essential` 探测之前, 节点不会接收客户端请求。如果配置被重载并且节点在此之前被认为健康, 则跳过 `persistent` 探测。如果没有这样的探测, 节点被认为是健康的。
- 如果为其配置的任何探测达到 `fails`, 或节点达到 `max_fails`, 则节点被认为是不健康的, 并且不会接收客户端请求。
- 要使不健康的节点再次被认为是健康的, 必须为其配置的所有探测达到各自的 `passes`; 之后, 还会考虑 `max_fails`。

## 内置变量

`http_upstream` 模块支持以下内置变量:

### `$upstream_probe (PRO)`

当前活动的 `upstream_probe` 的名称。

### `$upstream_probe_body (PRO)`

在一个 `upstream_probe` 中接收到的节点响应体; 其大小由 `max_body` 限制。

## 用户 ID

该模块设置适用于客户端识别的 Cookie。接收到的和设置的 Cookie 可以使用内置变量 `$uid_get` 和 `$uid_set` 进行记录。此模块兼容 Apache 的 `mod_uid` 模块。



## 配置示例

```
userid      on;
userid_name uid;
userid_domain example.com;
userid_path /;
userid_expires 365d;
userid_p3p  'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID";
```

## 指令

### userid

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>userid on   v1   log   off;</code> |
| 默认  | <code>userid off;</code>                 |
| 上下文 | http, server, location                   |

启用或禁用 Cookie 的设置以及接收 Cookie 的记录:

|     |                                  |
|-----|----------------------------------|
| on  | 启用版本 2 Cookie 的设置和接收 Cookie 的记录; |
| v1  | 启用版本 1 Cookie 的设置和接收 Cookie 的记录; |
| log | 禁用 Cookie 的设置, 但启用接收 Cookie 的记录; |
| off | 禁用 Cookie 的设置和接收 Cookie 的记录。     |

### userid\_domain

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>userid_domain name   none;</code> |
| 默认  | <code>userid_domain none;</code>        |
| 上下文 | http, server, location                  |

定义设置 Cookie 的域。参数 none 禁用为 Cookie 设置域。

### userid\_expires

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>userid_expires time   max   off;</code> |
| 默认  | <code>userid_expires off;</code>              |
| 上下文 | http, server, location                        |

设置浏览器应保留 Cookie 的时间。参数 `max` 将导致 Cookie 在“2037 年 12 月 31 日 23:55:55 GMT”过期。参数 `off` 将导致 Cookie 在浏览器会话结束时过期。

### userid\_flags

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>userid_flags off   flag ...;</code> |
| 默认  | <code>userid_flags off;</code>            |
| 上下文 | http, server, location                    |

如果参数不是 `off`, 则为 Cookie 定义一个或多个附加标志: `secure`、`httponly`、`samesite=strict`、`samesite=lax`、`samesite=none`。

### userid\_mark

|     |                                                    |
|-----|----------------------------------------------------|
| 语法  | <code>userid_mark letter   digit   =   off;</code> |
| 默认  | <code>userid_mark off;</code>                      |
| 上下文 | http, server, location                             |

如果参数不是 `off`, 启用 Cookie 标记机制并设置用作标记的字符。此机制用于在保留客户端标识符的同时添加或更改 `userid_p3p` 和/或 Cookie 过期时间。标记可以是英文字母 (区分大小写)、数字或“=” 字符。

如果设置了标记, 它将与 Cookie 中传递的客户端标识符的 base64 表示中的第一个填充符号进行比较。如果不匹配, Cookie 将使用指定的标记、过期时间和“P3P” 头重新发送。

### userid\_name

|     |                                |
|-----|--------------------------------|
| 语法  | <code>userid_name name;</code> |
| 默认  | <code>userid_name uid;</code>  |
| 上下文 | http, server, location         |

设置 Cookie 名称。

### userid\_p3p

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>userid_p3p string   none;</code> |
| 默认  | <code>userid_p3p none;</code>          |
| 上下文 | http, server, location                 |

为将随 Cookie 一起发送的“P3P”头字段设置一个值。如果指令设置为特殊值 `none`, 则“P3P”头不会在响应中发送。

### userid\_path

|     |                                |
|-----|--------------------------------|
| 语法  | <code>userid_path path;</code> |
| 默认  | <code>userid_path /;</code>    |
| 上下文 | http, server, location         |

定义设置 Cookie 的路径。

### userid\_service

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>userid_service number;</code>     |
| 默认  | <code>userid_service 服务器的 IP 地址;</code> |
| 上下文 | http, server, location                  |

如果标识符由多个服务器（服务）颁发, 则每个服务应分配自己的 `number` 以确保客户端标识符唯一。对于版本 1 的 Cookie, 默认值为零。对于版本 2 的 Cookie, 默认值是由服务器 IP 地址的最后四个八位字节组成的数字。

## 内置变量

### `$uid_got`

Cookie 名称和接收到的客户端标识符。

## \$uid\_reset

如果变量设置为非空字符串且不为 0, 则客户端标识符将被重置。特殊值 `log` 还会导致有关重置标识符的消息输出到 `error_log`。

## \$uid\_set

Cookie 名称和发送的客户端标识符。

## uWSGI

允许将请求传递给 uwsgi 服务器。

## 配置示例

```
location / {
    include uwsgi_params;
    uwsgi_pass localhost:9000;
}
```

## 指令

### uwsgi\_bind

|     |                                                      |
|-----|------------------------------------------------------|
| 语法  | <code>uwsgi_bind address [transparent]   off;</code> |
| 默认值 | —                                                    |
| 上下文 | http, server, location                               |

使到 uwsgi 服务器的外发连接从指定的本地 IP 地址和可选端口发起。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 `uwsgi_bind` 指令的效果, 允许系统自动分配本地 IP 地址和端口。

`transparent` 参数允许到 uwsgi 服务器的外发连接从非本地 IP 地址发起, 例如, 从客户端的真实 IP 地址发起:

```
uwsgi_bind $remote_addr transparent;
```

为了使此参数生效, 通常需要以超级用户 权限运行 Angie 工作进程。在 Linux 上不需要, 因为如果指定了 `transparent` 参数, 工作进程将从主进程继承 `CAP_NET_RAW` 能力。

**重要**

需要配置内核路由表以拦截来自 uwsgi 服务器的网络流量。

**uwsgi\_buffer\_size**

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>uwsgi_buffer_size size;</code>  |
| 默认值 | <code>uwsgi_buffer_size 4k 8k;</code> |
| 上下文 | http, server, location                |

设置用于读取从 uwsgi 服务器接收的响应第一部分的缓冲区大小。这部分通常包含一个小的响应头。默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 取决于平台。它可以变小。

**uwsgi\_buffering**

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>uwsgi_buffering size;</code> |
| 默认值 | <code>uwsgi_buffering on;</code>   |
| 上下文 | http, server, location             |

启用或禁用从 uwsgi 服务器接收的响应的缓冲。

|     |                                                                                                                                                                                                                                      |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| on  | Angie 尽快从 uwsgi 服务器接收响应, 将其保存在由 <code>uwsgi_buffer_size</code> 和 <code>uwsgi_buffers</code> 指令设置的缓冲区中。如果整个响应不适合内存, 可以将部分响应保存到磁盘上的临时文件中。写入临时文件由 <code>uwsgi_max_temp_file_size</code> 和 <code>uwsgi_temp_file_write_size</code> 指令控制。 |
| off | 响应同步传递给客户端, 立即接收。Angie 不会尝试从 uwsgi 服务器读取整个响应。Angie 从服务器一次可以接收的数据的最大大小由 <code>uwsgi_buffer_size</code> 指令设置。                                                                                                                          |

缓冲也可以通过在” X-Accel-Buffering” 响应头字段中传递” yes” 或” no” 来启用或禁用。可以使用 `uwsgi_ignore_headers` 指令禁用此功能。

### uwsgi\_buffers

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>uwsgi_buffers number size;</code> |
| 默认值 | <code>uwsgi_buffers 8 4k   8k;</code>   |
| 上下文 | http, server, location                  |

设置用于读取来自 uwsgi 服务器的响应的缓冲区的数量和大小, 针对单个连接。

默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 取决于平台。

### uwsgi\_busy\_buffers\_size

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>uwsgi_busy_buffers_size size;</code>     |
| 默认值 | <code>uwsgi_busy_buffers_size 8k   16k;</code> |
| 上下文 | http, server, location                         |

当启用从 uwsgi 服务器接收的响应的缓冲时, 限制在响应尚未完全读取时可以繁忙地向客户端发送响应的缓冲区的总大小。与此同时, 其余缓冲区可以用于读取响应, 如果需要, 将响应的一部分缓冲到临时文件中。

默认情况下, 大小由 `uwsgi_buffer_size` 和 `uwsgi_buffers` 指令设置的两个缓冲区的大小限制。

### uwsgi\_cache

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>uwsgi_cache zone   off;</code> |
| 默认值 | <code>uwsgi_cache off;</code>        |
| 上下文 | http, server, location               |

定义用于缓存的共享内存区。相同的区域可以在多个地方使用。参数值可以包含变量。

|     |                 |
|-----|-----------------|
| off | 禁用从先前配置级别继承的缓存。 |
|-----|-----------------|

### uwsgi\_cache\_background\_update

|     |                                                      |
|-----|------------------------------------------------------|
| 语法  | <code>uwsgi_cache_background_update on   off;</code> |
| 默认值 | <code>uwsgi_cache_background_update off;</code>      |
| 上下文 | http, server, location                               |

允许启动后台子请求来更新过期的缓存项, 同时将陈旧的缓存响应返回给客户端。

#### ⚠ 注意

请注意, 必须允许 在更新时使用陈旧的缓存响应。

### uwsgi\_cache\_bypass

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>uwsgi_cache_bypass ...;</code> |
| 默认值 | —                                    |
| 上下文 | http, server, location               |

定义在什么条件下不会从缓存中获取响应。如果字符串参数的至少一个值不为空且不等于”0”, 则不会从缓存中获取响应:

```
uwsgi_cache_bypass $cookie_nocache $arg_nocache$arg_comment;  
uwsgi_cache_bypass $http_pragma $http_authorization;
```

可以与 `uwsgi_no_cache` 指令一起使用。

### uwsgi\_cache\_key

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>uwsgi_cache_key string;</code> |
| 默认值 | —                                    |
| 上下文 | http, server, location               |

定义缓存的键, 例如

```
uwsgi_cache_key localhost:9000$request_uri;
```

### uwsgi\_cache\_lock

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>uwsgi_cache_lock on   off;</code> |
| 默认值 | <code>uwsgi_cache_lock off;</code>      |
| 上下文 | http, server, location                  |

启用时, 同一时间将只允许一个请求通过向 uwsgi 服务器发送请求来填充一个新缓存元素, 按照 `uwsgi_cache_key` 指令识别。相同缓存元素的其他请求将等待响应出现在缓存中或该元素的缓存锁被释放, 最长等待时间由 `uwsgi_cache_lock_timeout` 指令设置。

### uwsgi\_cache\_lock\_age

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>uwsgi_cache_lock_age time;</code> |
| 默认值 | <code>uwsgi_cache_lock_age 5s;</code>   |
| 上下文 | http, server, location                  |

如果最后发送到 uwsgi 服务器的请求在指定时间内未完成, 可以再向 uwsgi 服务器发送一个请求。

### uwsgi\_cache\_lock\_timeout

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>uwsgi_cache_lock_timeout time;</code> |
| 默认值 | <code>uwsgi_cache_lock_timeout 5s;</code>   |
| 上下文 | http, server, location                      |

设置 `uwsgi_cache_lock` 的超时时间。当时间到期时, 请求将被传递给 uwsgi 服务器, 但响应不会被缓存。

### uwsgi\_cache\_max\_range\_offset

|     |                                                   |
|-----|---------------------------------------------------|
| 语法  | <code>uwsgi_cache_max_range_offset number;</code> |
| 默认值 | —                                                 |
| 上下文 | http, server, location                            |

为字节范围请求设置一个字节偏移量。如果范围超出偏移量, 范围请求将被传递给 uwsgi 服务器, 响应不会被缓存。



### uwsgi\_cache\_methods

|     |                                                         |
|-----|---------------------------------------------------------|
| 语法  | <code>uwsgi_cache_methods GET   HEAD   POST ...;</code> |
| 默认值 | <code>uwsgi_cache_methods GET HEAD;</code>              |
| 上下文 | http, server, location                                  |

如果客户端请求方法在此指令中列出, 则响应将被缓存。“GET” 和“HEAD” 方法总是添加到列表中, 但建议显式指定它们。另见 `uwsgi_no_cache` 指令。

### uwsgi\_cache\_min\_uses

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>uwsgi_cache_min_uses number;</code> |
| 默认值 | <code>uwsgi_cache_min_uses 1;</code>      |
| 上下文 | http, server, location                    |

设置响应将被缓存的请求次数。

### uwsgi\_cache\_path

|     |                                                                                                                                                                                                                                                                                                                                |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>uwsgi_cache_path path [levels=levels] [use_temp_path=on   off]<br/>         keys_zone=name:size [inactive=time] [max_size=size] [min_free=size]<br/>         [manager_files=number] [manager_sleep=time] [manager_threshold=time]<br/>         [loader_files=number] [loader_sleep=time] [loader_threshold=time];</code> |
| 默认值 | —                                                                                                                                                                                                                                                                                                                              |
| 上下文 | http                                                                                                                                                                                                                                                                                                                           |

设置缓存的路径和其他参数。缓存数据存储在文件中。缓存中的文件名是应用 MD5 函数到缓存键的结果。

`levels` 参数定义缓存的层级结构: 从 1 到 3, 每个层级接受值 1 或 2。例如, 在以下配置中:

```
uwsgi_cache_path /data/angie/cache levels=1:2 keys_zone=one:10m;
```

缓存中的文件名将如下所示:

```
/data/angie/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

缓存的响应首先写入临时文件, 然后重命名该文件。临时文件和缓存可以放置在不同的文件系统上。但是, 请注意, 在这种情况下, 文件是跨两个文件系统复制的, 而不是廉价的重命名操作。因此, 建议在任意给定位置将缓存和存放临时文件的目录放在同一个文件系统中。

临时文件的目录是基于 `use_temp_path` 参数设置的。

|     |                                                                           |
|-----|---------------------------------------------------------------------------|
| on  | 如果此参数被省略或设置为 on, 则将使用给定 <i>location</i> 的 <i>uwsgi_temp_path</i> 指令设置的目录。 |
| off | 临时文件将直接放在缓存目录中。                                                           |

此外, 所有活动的键和数据的信息都存储在共享内存区域中, 其名称和大小由 *keys\_zone* 参数配置。一个兆字节的区域可以存储大约 8000 个键。

在 *inactive* 参数指定的时间内未访问的缓存数据将从缓存中删除, 而不考虑其新鲜度。

默认情况下, *inactive* 设置为 10 分钟。

一个特殊的 **缓存管理** 进程监控最大缓存大小和文件系统上的最小可用空间, 当超出大小或可用空间不足时, 它会删除最近最少使用的数据。数据在迭代中删除。

|                          |                                |
|--------------------------|--------------------------------|
| <i>max_size</i>          | 最大 <i>uwsgi_cache</i> 大小       |
| <i>min_free</i>          | 文件系统上缓存的最小可用空间                 |
| <i>manager_files</i>     | 限制在一次迭代中要删除的项目数量<br>默认情况下, 100 |
| <i>manager_threshold</i> | 限制一次迭代的持续时间<br>默认情况下, 200 毫秒   |
| <i>manager_sleep</i>     | 配置交互之间的间隔<br>默认情况下, 50 毫秒      |

Angie 启动一分钟后, 特殊的 **缓存加载器** 进程被激活。它将文件系统上存储的以前缓存的数据的信息加载到缓存区域中。加载也是在迭代中完成的。

|                         |                                |
|-------------------------|--------------------------------|
| <i>loader_files</i>     | 限制在一次迭代中要加载的项目数量<br>默认情况下, 100 |
| <i>loader_threshold</i> | 限制一次迭代的持续时间<br>默认情况下, 200 毫秒   |
| <i>loader_sleep</i>     | 配置交互之间的间隔<br>默认情况下, 50 毫秒      |

### uwsgi\_cache\_revalidate

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>uwsgi_cache_revalidate on   off;</code> |
| 默认值 | <code>uwsgi_cache_revalidate off;</code>      |
| 上下文 | http, server, location                        |

启用使用 "If-Modified-Since" 和 "If-None-Match" 头字段的条件请求重新验证过期的缓存项目。

## uwsgi\_cache\_use\_stale

|     |                                                                                                                                     |
|-----|-------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | uwsgi_cache_use_stale error   timeout   invalid_header   updating   http_500   http_503   http_403   http_404   http_429   off ...; |
| 默认值 | uwsgi_cache_use_stale off;                                                                                                          |
| 上下文 | http, server, location                                                                                                              |

确定在与 uwsgi 服务器通信时可以使用过期缓存响应的情况。该指令的参数与 *uwsgi\_next\_upstream* 指令的参数相匹配。

|          |                                                             |
|----------|-------------------------------------------------------------|
| error    | 如果无法选择 uwsgi 服务器来处理请求, 则允许使用过期的缓存响应。                        |
| updating | 额外参数, 允许在缓存响应正在更新时使用过期的缓存响应。这允许在更新缓存数据时最小化对 uwsgi 服务器的访问次数。 |

使用过期的缓存响应也可以直接在响应头中启用, 指定响应过期后的一段秒数:

- "Cache-Control" 头字段的 *stale-while-revalidate* 扩展允许在缓存响应正在更新时使用过期的缓存响应。
- "Cache-Control" 头字段的 *stale-if-error* 扩展允许在发生错误时使用过期的缓存响应。

### 备注

这优先级低于使用指令参数。

为了在填充新缓存元素时最小化对 uwsgi 服务器的访问次数, 可以使用 *uwsgi\_cache\_lock* 指令。

## uwsgi\_cache\_valid

|         |                                                     |
|---------|-----------------------------------------------------|
| Syntax  | uwsgi_cache_valid [ <i>code ...</i> ] <i>time</i> ; |
| Default | —                                                   |
| Context | http, server, location                              |

为不同的响应代码设置缓存时间。例如, 以下指令

```
uwsgi_cache_valid 200 302 10m;
uwsgi_cache_valid 404      1m;
```

为代码 200 和 302 的响应设置 10 分钟的缓存时间, 为代码 404 的响应设置 1 分钟。

如果只指定缓存时间

```
uwsgi_cache_valid 5m;
```

则仅缓存 200、301 和 302 响应。

此外, 可以指定 `any` 参数以缓存任何响应:

```
uwsgi_cache_valid 200 302 10m;  
uwsgi_cache_valid 301      1h;  
uwsgi_cache_valid any      1m;
```

### **i** 备注

缓存参数也可以直接在响应头中设置。这优先级高于使用指令设置的缓存时间

- "X-Accel-Expires" 头字段以秒为单位设置响应的缓存时间。零值禁用响应的缓存。如果值以 @ 前缀开头, 则设置自 Epoch 以来的秒数绝对时间, 直到响应可能被缓存。
- 如果头中不包含 "X-Accel-Expires" 字段, 缓存参数可以在头字段 "Expires" 或 "Cache-Control" 中设置。
- 如果头中包含 "Set-Cookie" 字段, 则该响应将不被缓存。
- 如果头中包含值为 "\*" 的 "Vary" 字段, 则该响应将不被缓存。如果头中包含其他值的 "Vary" 字段, 则该响应将被缓存, 并考虑相应请求头字段。

可以使用 `uwsgi_ignore_headers` 指令禁用对这些响应头字段之一或多个的处理。

## uwsgi\_connect\_timeout

|         |                                          |
|---------|------------------------------------------|
| Syntax  | <code>uwsgi_connect_timeout time;</code> |
| Default | <code>uwsgi_connect_timeout 60s;</code>  |
| Context | http, server, location                   |

定义与 uwsgi 服务器建立连接的超时时间。应注意, 此超时时间通常不能超过 75 秒。

## uwsgi\_connection\_drop

|         |                                                     |
|---------|-----------------------------------------------------|
| Syntax  | <code>uwsgi_connection_drop time   on   off;</code> |
| Default | <code>uwsgi_connection_drop off;</code>             |
| Context | http, server, location                              |

启用在代理服务器从组中移除或被 `eresolve` 进程或 `API command DELETE` 标记为永久不可用后终止所有连接。

当下一个读或写事件在客户端或代理服务器中处理时, 连接被终止。

设置 *time* 启用连接终止 timeout; 设置 *on*, 连接立即断开。

### uwsgi\_force\_ranges

|         |                                      |
|---------|--------------------------------------|
| Syntax  | <code>uwsgi_force_ranges off;</code> |
| Default | <code>uwsgi_force_ranges off;</code> |
| Context | http, server, location               |

启用对来自 uwsgi 服务器的缓存和未缓存响应的字节范围支持, 而不考虑这些响应中的"Accept-Ranges" 字段。

### uwsgi\_hide\_header

|         |                                              |
|---------|----------------------------------------------|
| Syntax  | <code>uwsgi_hide_header <i>field</i>;</code> |
| Default | —                                            |
| Context | http, server, location                       |

默认情况下, Angie 不会将 uwsgi 服务器响应中的"Status" 和"X-Accel-..." 头字段传递给客户端。*uwsgi\_hide\_header* 指令设置不会传递的额外字段。如果需要允许传递字段, 可以使用 *uwsgi\_pass\_header* 指令。

### uwsgi\_ignore\_client\_abort

|         |                                                  |
|---------|--------------------------------------------------|
| Syntax  | <code>uwsgi_ignore_client_abort on   off;</code> |
| Default | <code>uwsgi_ignore_client_abort off;</code>      |
| Context | http, server, location                           |

确定当客户端在未等待响应的情况下关闭连接时, 是否应关闭与 uwsgi 服务器的连接。

### uwsgi\_ignore\_headers

|         |                                                     |
|---------|-----------------------------------------------------|
| Syntax  | <code>uwsgi_ignore_headers <i>field</i> ...;</code> |
| Default | —                                                   |
| Context | http, server, location                              |

禁用对来自 uwsgi 服务器的某些响应头字段的处理。可以忽略以下字段: "X-Accel-Redirect"、"X-Accel-Expires"、"X-Accel-Limit-Rate"、"X-Accel-Buffering"、"X-Accel-Charset"、"Expires"、"Cache-Control"、"Set-Cookie" 和 "Vary"。

如果未禁用, 对这些头字段的处理将产生以下效果:

- "X-Accel-Expires"、"Expires"、"Cache-Control"、"Set-Cookie" 和 "Vary" 设置响应 *caching* 的参数;
- "X-Accel-Redirect" 执行 *internal* 重定向到指定的 URI;
- "X-Accel-Limit-Rate" 设置将响应传输给客户端的 *rate limit*;
- "X-Accel-Buffering" 启用或禁用响应的 *buffering*;
- "X-Accel-Charset" 设置响应的所需 *charset*。

### uwsgi\_intercept\_errors

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>uwsgi_intercept_errors on   off;</code> |
| 默认值 | <code>uwsgi_intercept_errors off;</code>      |
| 上下文 | http, server, location                        |

决定是否应将 uwsgi 服务器响应的代码大于或等于 300 的情况传递给客户端, 或者拦截并重定向到 Angie 进行使用 *error\_page* 指令的处理。

### uwsgi\_limit\_rate

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>uwsgi_limit_rate rate;</code> |
| 默认值 | <code>uwsgi_limit_rate 0;</code>    |
| 上下文 | http, server, location              |

限制从 uwsgi 服务器读取响应的速度。 *rate* 以每秒字节数指定, 并且可以包含变量。

|   |        |
|---|--------|
| 0 | 禁用速率限制 |
|---|--------|

#### 备注

该限制是针对每个请求设置的, 因此如果 Angie 同时打开两个到 uwsgi 服务器的连接, 总的速率将是指定限制的两倍。 仅在启用了对 uwsgi 服务器响应的 *buffering* 时, 限制才有效。

### uwsgi\_max\_temp\_file\_size

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>uwsgi_max_temp_file_size size;</code>  |
| 默认值 | <code>uwsgi_max_temp_file_size 1024m;</code> |
| 上下文 | <code>http, server, location</code>          |

当启用了 `uwsgi` 服务器响应的 *buffering*, 并且整个响应不适合由 `uwsgi_buffer_size` 和 `uwsgi_buffers` 指令设置的缓冲区时, 可以将响应的一部分保存到临时文件中。此指令设置临时文件的最大大小。一次写入临时文件的数据大小由 `uwsgi_temp_file_write_size` 指令设置。

|   |              |
|---|--------------|
| 0 | 禁用响应到临时文件的缓冲 |
|---|--------------|

#### 备注

此限制不适用于将被缓存或存储在磁盘上的响应。

### uwsgi\_modifier1

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>uwsgi_modifier1 size;</code>  |
| 默认值 | <code>uwsgi_modifier1 0;</code>     |
| 上下文 | <code>http, server, location</code> |

设置 `uwsgi` 数据包头中 `modifier1` 字段的值。

### uwsgi\_modifier2

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>uwsgi_modifier2 size;</code>  |
| 默认值 | <code>uwsgi_modifier2 0;</code>     |
| 上下文 | <code>http, server, location</code> |

设置 `uwsgi` 数据包头中 `modifier2` 字段的值。

## uwsgi\_next\_upstream

|     |                                                                                                                                         |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | uwsgi_next_upstream error   timeout   invalid_header   http_500   http_503   http_403   http_404   http_429   non_idempotent   off ...; |
| 默认值 | uwsgi_next_upstream error timeout;                                                                                                      |
| 上下文 | http, server, location                                                                                                                  |

指定在何种情况下应将请求传递给上游池中的下一个服务器:

|                |                                                                                         |
|----------------|-----------------------------------------------------------------------------------------|
| error          | 在与服务器建立连接、传递请求或读取响应头时发生了错误;                                                             |
| timeout        | 在与服务器建立连接、传递请求或读取响应头时发生了超时;                                                             |
| invalid_header | 服务器返回了空或无效的响应;                                                                          |
| http_500       | 服务器返回代码 500 的响应;                                                                        |
| http_503       | 服务器返回代码 503 的响应;                                                                        |
| http_403       | 服务器返回代码 403 的响应;                                                                        |
| http_404       | 服务器返回代码 404 的响应;                                                                        |
| http_429       | 服务器返回代码 429 的响应;                                                                        |
| non_idempotent | 通常, 具有 <b>非幂等</b> 方法 (POST, LOCK, PATCH) 的请求在请求已被发送到上游服务器后不会传递给下一个服务器; 启用此选项显式允许重试此类请求; |
| off            | 禁用将请求传递给下一个服务器。                                                                         |

### 备注

需要注意的是, 只有在尚未将任何数据发送给客户端时, 才可以将请求传递给下一个服务器。也就是说, 如果在响应传输过程中发生错误或超时, 则无法修复。

该指令还定义了与服务器通信的**不成功尝试**。

|                |                          |
|----------------|--------------------------|
| error          | 始终被视为不成功的尝试, 即使它们未在指令中指定 |
| timeout        |                          |
| invalid_header |                          |
| http_500       | 仅当在指令中指定时才被视为不成功的尝试      |
| http_503       |                          |
| http_429       |                          |
| http_403       | 永远不会被视为不成功的尝试            |
| http_404       |                          |

将请求传递给下一个服务器可以通过**尝试次数**和**时间**来限制。



### uwsgi\_next\_upstream\_timeout

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>uwsgi_next_upstream_timeout time;</code> |
| 默认值 | <code>uwsgi_next_upstream_timeout 0;</code>    |
| 上下文 | http, server, location                         |

限制在请求可以传递给下一个服务器的时间。

|   |       |
|---|-------|
| 0 | 关闭此限制 |
|---|-------|

### uwsgi\_next\_upstream\_tries

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>uwsgi_next_upstream_tries number;</code> |
| 默认值 | <code>uwsgi_next_upstream_tries 0;</code>      |
| 上下文 | http, server, location                         |

限制将请求传递给下一个服务器的可能尝试次数。

|   |       |
|---|-------|
| 0 | 关闭此限制 |
|---|-------|

### uwsgi\_no\_cache

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>uwsgi_no_cache string ...;</code> |
| 默认值 | —                                       |
| 上下文 | http, server, location                  |

定义在何种条件下响应不会被保存到缓存中。如果字符串参数的至少一个值不为空且不等于“0”，则响应不会被保存：

```
uwsgi_no_cache $cookie_nocache $arg_nocache$arg_comment;  
uwsgi_no_cache $http_pragma $http_authorization;
```

可以与 `uwsgi_cache_bypass` 指令一起使用。

## uwsgi\_param

|     |                                                          |
|-----|----------------------------------------------------------|
| 语法  | <code>uwsgi_param parameter value [if_not_empty];</code> |
| 默认值 | —                                                        |
| 上下文 | http, server, location                                   |

设置应传递给 uwsgi 服务器的参数。该值可以包含文本、变量及其组合。如果且仅在当前级别未定义 uwsgi\_param 指令时, 这些指令会从先前的配置级别继承。

标准 CGI 环境变量 应作为 uwsgi 头提供, 参见分发中提供的 `uwsgi_params` 文件:

```
location / {  
    include uwsgi_params;  
    # ...  
}
```

如果指令使用 `if_not_empty` 指定, 则仅在其值不为空时才将该参数传递给服务器:

```
uwsgi_param HTTPS $https if_not_empty;
```

## uwsgi\_pass

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>uwsgi_pass [protocol://] address;</code> |
| 默认值 | —                                              |
| 上下文 | location, if in location                       |

设置 uwsgi 服务器的协议和地址, 以及一个可选的 URI, 将位置映射到该地址。作为协议, 可以指定 `uwsgi` 或 `suwsgi` (安全的 uwsgi, 通过 SSL 的 uwsgi)。地址可以指定为域名或 IP 地址以及端口:

```
uwsgi_pass localhost:9000;  
uwsgi_pass uwsgi://localhost:9000;  
uwsgi_pass suwsgi://[2001:db8::1]:9090;
```

或作为 UNIX 域套接字路径:

```
uwsgi_pass unix:/tmp/uwsgi.socket;
```

如果域名解析为多个地址, 则所有地址将以轮询方式使用。此外, 地址可以指定为服务器组。如果使用组, 则不能为其指定端口; 而是为组内的每个服务器分别指定端口。

参数值可以包含变量。在这种情况下, 如果地址指定为域名, 则在描述的服务器组中搜索该名称, 如果未找到, 则使用解析器 确定。

### uwsgi\_pass\_header

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>uwsgi_pass_header field ...;</code> |
| 默认值 | —                                         |
| 上下文 | http, server, location                    |

允许从 uwsgi 服务器传递否则被禁用的 头字段到客户端。

### uwsgi\_pass\_request\_body

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>uwsgi_pass_request_body on   off;</code> |
| 默认值 | <code>uwsgi_pass_request_body on;</code>       |
| 上下文 | http, server, location                         |

指示是否将原始请求主体传递给 uwsgi 服务器。另请参见 `uwsgi_pass_request_headers` 指令。

### uwsgi\_pass\_request\_headers

|     |                                                   |
|-----|---------------------------------------------------|
| 语法  | <code>uwsgi_pass_request_headers on   off;</code> |
| 默认值 | <code>uwsgi_pass_request_headers on;</code>       |
| 上下文 | http, server, location                            |

指示是否将原始请求的头字段传递给 uwsgi 服务器。另请参见 `uwsgi_pass_request_body` 指令。

### uwsgi\_read\_timeout

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>uwsgi_read_timeout time;</code> |
| 默认值 | <code>uwsgi_read_timeout 60s;</code>  |
| 上下文 | http, server, location                |

定义从 uwsgi 服务器读取响应的超时时间。超时时间仅在两次连续读取操作之间设置，而不是用于整个响应的传输。如果 uwsgi 服务器在此时间内没有传输任何内容，则连接将关闭。

### uwsgi\_request\_buffering

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>uwsgi_request_buffering on   off;</code> |
| 默认值 | <code>uwsgi_request_buffering on;</code>       |
| 上下文 | http, server, location                         |

启用或禁用客户端请求主体的缓冲。

|     |                                                                      |
|-----|----------------------------------------------------------------------|
| on  | 在将请求发送到 uwsgi 服务器之前, 从客户端读取整个请求主体。                                   |
| off | 请求主体在接收到时立即发送到 uwsgi 服务器。在这种情况下, 如果 Angie 已经开始发送请求主体, 请求无法传递给下一个服务器。 |

当使用 HTTP/1.1 分块传输编码发送原始请求主体时, 无论指令值如何, 请求主体都将被缓冲。

### uwsgi\_send\_timeout

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>uwsgi_send_timeout time;</code> |
| 默认值 | <code>uwsgi_send_timeout 60s;</code>  |
| 上下文 | http, server, location                |

设置向 uwsgi 服务器传输请求的超时时间。超时时间仅在两次连续写入操作之间设置, 而不是用于整个请求的传输。如果 uwsgi 服务器在此时间内没有接收到任何内容, 则连接将关闭。

### uwsgi\_socket\_keepalive

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>uwsgi_socket_keepalive on   off;</code> |
| 默认值 | <code>uwsgi_socket_keepalive off;</code>      |
| 上下文 | http, server, location                        |

配置到 uwsgi 服务器的传出连接的“TCP 保持活动”行为。

|    |                                         |
|----|-----------------------------------------|
| "" | 默认为操作系统的设置对套接字生效。                       |
| on | 为套接字启用 <code>SO_KEEPALIVE</code> 套接字选项。 |

### uwsgi\_ssl\_certificate

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>uwsgi_ssl_certificate file;</code> |
| 默认值 | —                                        |
| 上下文 | http, server, location                   |

指定用于认证到安全 uwsgi 服务器的 PEM 格式证书文件。文件名中可以使用变量。

### uwsgi\_ssl\_certificate\_key

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>uwsgi_ssl_certificate_key file;</code> |
| 默认值 | —                                            |
| 上下文 | http, server, location                       |

指定用于认证到安全 uwsgi 服务器的 PEM 格式秘密密钥文件。

可以指定 “engine:name:id” 而不是文件，从 OpenSSL 引擎名称加载具有指定 id 的秘密密钥。文件名中可以使用变量。

### uwsgi\_ssl\_ciphers

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>uwsgi_ssl_ciphers ciphers;</code> |
| 默认值 | <code>uwsgi_ssl_ciphers DEFAULT;</code> |
| 上下文 | http, server, location                  |

指定请求到安全 uwsgi 服务器时启用的密码。密码以 OpenSSL 库理解的格式指定。

可以使用 “`openssl ciphers`” 命令查看完整列表。

### uwsgi\_ssl\_conf\_command

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>uwsgi_ssl_conf_command name value;</code> |
| 默认值 | —                                               |
| 上下文 | http, server, location                          |

在与安全 uwsgi 服务器建立连接时设置任意 OpenSSL 配置 ‘命令 <[https://www.openssl.org/docs/man1.1.1/man3/SSL\\_CONF\\_cmd.html](https://www.openssl.org/docs/man1.1.1/man3/SSL_CONF_cmd.html)>’。

**重要**

使用 OpenSSL 1.0.2 或更高版本时支持该指令。

可以在同一级别指定多个 `uwsgi_ssl_conf_command` 指令。如果且仅当当前级别没有定义 `uwsgi_ssl_conf_command` 指令时, 这些指令将从上一级配置继承。

**小心**

请注意, 直接配置 OpenSSL 可能导致意外行为。

**uwsgi\_ssl\_crl**

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>uwsgi_ssl_crl file;</code> |
| 默认值 | —                                |
| 上下文 | http, server, location           |

指定用于验证安全 uwsgi 服务器证书的 PEM 格式撤销证书 (CRL) 文件。

**uwsgi\_ssl\_name**

|     |                                                     |
|-----|-----------------------------------------------------|
| 语法  | <code>uwsgi_ssl_name name;</code>                   |
| 默认值 | <code>uwsgi_ssl_name `host from uwsgi_pass`;</code> |
| 上下文 | http, server, location                              |

允许覆盖用于验证安全 uwsgi 服务器证书的服务器名称, 并在与安全 uwsgi 服务器建立连接时通过 SNI 传递。默认情况下, 使用 `uwsgi_pass` URL 的主机部分。

**uwsgi\_ssl\_password\_file**

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>uwsgi_ssl_password_file file;</code> |
| 默认值 | —                                          |
| 上下文 | http, server, location                     |

指定用于秘密密钥的密码短语文件, 每个密码短语在单独的行中指定。在加载密钥时将依次尝试这些密码短语。

### uwsgi\_ssl\_protocols

|     |                                                                                         |
|-----|-----------------------------------------------------------------------------------------|
| 语法  | <code>uwsgi_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];</code> |
| 默认值 | <code>uwsgi_ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;</code>                         |
| 上下文 | <code>http, server, location</code>                                                     |

在 1.2.0 版本发生变更: TLSv1.3 参数已添加到默认集。

启用请求到安全 uwsgi 服务器时指定的协议。

### uwsgi\_ssl\_server\_name

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>uwsgi_ssl_server_name on   off;</code> |
| 默认值 | <code>uwsgi_ssl_server_name off;</code>      |
| 上下文 | <code>http, server, location</code>          |

启用或禁用通过 服务器名称指示 TLS 扩展 (SNI, RFC 6066) 在与安全 uwsgi 服务器建立连接时传递由 `uwsgi_ssl_name` 指令设置的服务器名称。

### uwsgi\_ssl\_session\_reuse

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>uwsgi_ssl_session_reuse on   off;</code> |
| 默认值 | <code>uwsgi_ssl_session_reuse on;</code>       |
| 上下文 | <code>http, server, location</code>            |

确定与 uwsgi 服务器工作时是否可以重用 SSL 会话。如果日志中出现 “`SSL3_GET_FINISHED:digest check failed`” 错误, 请尝试禁用会话重用。

### uwsgi\_ssl\_trusted\_certificate

|     |                                                         |
|-----|---------------------------------------------------------|
| 语法  | <code>uwsgi_ssl_trusted_certificate <i>file</i>;</code> |
| 默认值 | —                                                       |
| 上下文 | <code>http, server, location</code>                     |

指定用于验证 安全 uwsgi 服务器证书的 PEM 格式可信 CA 证书文件。

### uwsgi\_ssl\_verify

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>uwsgi_ssl_verify on   off;</code> |
| 默认值 | <code>uwsgi_ssl_verify off;</code>      |
| 上下文 | <code>http, server, location</code>     |

启用或禁用对安全 uwsgi 服务器证书的验证。

### uwsgi\_ssl\_verify\_depth

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>uwsgi_ssl_verify_depth number;</code> |
| 默认值 | <code>uwsgi_ssl_verify_depth 1;</code>      |
| 上下文 | <code>http, server, location</code>         |

设置安全 uwsgi 服务器证书链的验证深度。

### uwsgi\_store

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>uwsgi_store on   off   string;</code> |
| 默认值 | <code>uwsgi_store off;</code>               |
| 上下文 | <code>http, server, location</code>         |

启用将文件保存到磁盘。

|                  |                                                         |
|------------------|---------------------------------------------------------|
| <code>on</code>  | 保存具有与指令 <code>alias</code> 或 <code>root</code> 相对应路径的文件 |
| <code>off</code> | 禁用文件保存                                                  |

可以使用带变量的 `string` 显式设置文件名：

```
uwsgi_store /data/www$original_uri;
```

文件的修改时间根据接收到的“Last-Modified”响应头字段设置。响应首先写入一个临时文件，然后将文件重命名。临时文件和持久存储可以放在不同的文件系统上。然而，请注意，这种情况下文件是在两个文件系统之间复制，而不是简单的重命名操作。因此，建议对于任何给定的 `location`，保存的文件和由 `uwsgi_temp_path` 指令设置的临时文件目录，都放在同一个文件系统上。

此指令可用于创建静态不可变文件的本地副本，例如：

```
location /images/ {  
    root /data/www;
```



```

    error_page      404 = /fetch$uri;
}

location /fetch/ {
    internal;

    uwsgi_pass      backend:9000;
    ...

    uwsgi_store     on;
    uwsgi_store_access user:rw group:rw all:r;
    uwsgi_temp_path /data/temp;

    alias           /data/www/;
}

```

### uwsgi\_store\_access

|     |                                                        |
|-----|--------------------------------------------------------|
| 语法  | <code>uwsgi_store_access users:permissions ...;</code> |
| 默认值 | <code>uwsgi_store_access user:rw;</code>               |
| 上下文 | http, server, location                                 |

设置新创建文件和目录的访问权限, 例如:

```
uwsgi_store_access user:rw group:rw all:r;
```

如果指定了任何 group 或 all 访问权限, 则用户权限可以省略:

```
uwsgi_store_access group:rw all:r;
```

### uwsgi\_temp\_file\_write\_size

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>uwsgi_temp_file_write_size size;</code>   |
| 默认值 | <code>uwsgi_temp_file_write_size 8k 16k;</code> |
| 上下文 | http, server, location                          |

限制在启用从 uwsgi 服务器到临时文件的响应缓冲时, 一次写入临时文件的数据大小。默认情况下, 大小由 `uwsgi_buffer_size` 和 `uwsgi_buffers` 指令设置的两个缓冲区限制。临时文件的最大大小由 `uwsgi_max_temp_file_size` 指令设置。

## uwsgi\_temp\_path

|         |                                                                                                   |
|---------|---------------------------------------------------------------------------------------------------|
| Syntax  | <code>uwsgi_temp_path path [level1 [level2 [level3]]];</code>                                     |
| Default | <code>uwsgi_temp_path uwsgi_temp;</code> (路径依赖于 <code>--http-uwsgi-temp-path</code> build option) |
| Context | http, server, location                                                                            |

定义一个目录, 用于存储从 uwsgi 服务器接收到的数据的临时文件。最多可以在指定的目录下使用三层子目录结构。例如, 在以下配置中

```
uwsgi_temp_path /spool/angie/uwsgi_temp 1 2;
```

一个临时文件可能看起来像这样:

```
/spool/angie/uwsgi_temp/7/45/00000123457
```

另请参见 `uwsgi_cache_path` 指令的 `use_temp_path` 参数。

## HTTP/2

提供对 ‘HTTP/2 <<https://datatracker.ietf.org/doc/html/rfc9113>>’ 的支持。

当从源代码构建时, 此模块默认不构建; 应通过 `--with-http_v2_module` 构建选项启用。

在来自我们仓库的包和镜像中, 该模块包含在构建中。

## 配置示例

```
server {  
    listen 443 ssl;  
  
    http2 on;  
  
    ssl_certificate server.crt;  
    ssl_certificate_key server.key;  
}
```

### 重要

请注意, 通过 TLS 接受 HTTP/2 连接需要“应用层协议协商”(ALPN) TLS 扩展支持, 该支持自 OpenSSL 1.0.2 版本起可用。

如果 `ssl_prefer_server_ciphers` 指令设置为“on”, 则 `ciphers` 应配置为符合 RFC 9113, 附录 A 黑名单并被客户端支持。

## 指令

### http2

Added in version 1.2.0.

|     |                              |
|-----|------------------------------|
| 语法  | <code>http2 on   off;</code> |
| 默认  | <code>http2 off;</code>      |
| 上下文 | <code>http, server</code>    |

启用 ‘HTTP/2 <<https://datatracker.ietf.org/doc/html/rfc9113>>’ \_\_\_ 协议。

### http2\_body\_preread\_size

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>http2_body_preread_size size;</code> |
| 默认  | —                                          |
| 上下文 | <code>http, server</code>                  |

设置每个请求的缓冲区大小, 在请求体开始处理前可以存储请求体。

### http2\_chunk\_size

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>http2_chunk_size size;</code> |
| 默认  | <code>http2_chunk_size 8k;</code>   |
| 上下文 | <code>http, server, location</code> |

设置响应体切片的最大块大小。过低的值会导致更高的开销。过高的值会因 ‘HOL 阻塞 <[http://en.wikipedia.org/wiki/Head-of-line\\_blocking](http://en.wikipedia.org/wiki/Head-of-line_blocking)>’ \_\_\_ 而降低优先级。

### http2\_max\_concurrent\_pushes

自 1.2.0 版本弃用。

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>http2_max_concurrent_pushes number;</code> |
| 默认  | <code>http2_max_concurrent_pushes 10;</code>     |
| 上下文 | <code>http, server</code>                        |

限制连接中并发推送 请求的最大数量。

### http2\_max\_concurrent\_streams

|     |                                                   |
|-----|---------------------------------------------------|
| 语法  | <code>http2_max_concurrent_streams number;</code> |
| 默认  | <code>http2_max_concurrent_streams 128;</code>    |
| 上下文 | <code>http, server</code>                         |

设置连接中并发 HTTP/2 流的最大数量。

### http2\_push

自 1.2.0 版本弃用。

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>http2_push uri   off;</code>  |
| 默认  | <code>http2_push off;</code>        |
| 上下文 | <code>http, server, location</code> |

主动发送 (推送) 请求到指定的 `uri`, 并与原始请求的响应一起发送。只有具有绝对路径的相对 URI 将被处理, 例如:

```
http2_push /static/css/main.css;
```

`uri` 值可以包含变量。

在同一配置级别可以指定多个 `http2_push` 指令。 `off` 参数取消从上一级配置继承的 `http2_push` 指令的效果。

### http2\_push\_preload

自 1.2.0 版本弃用。

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>http2_push_preload on   off;</code> |
| 默认  | <code>http2_push_preload off;</code>      |
| 上下文 | <code>http, server, location</code>       |

启用在 “Link” 响应头字段中指定的 [预加载链接](#) 自动转换为 [推送](#) 请求。

## http2\_recv\_buffer\_size

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>http2_recv_buffer_size size;</code> |
| 默认  | <code>http2_recv_buffer_size 256k;</code> |
| 上下文 | <code>http</code>                         |

设置每个 *worker* 输入缓冲区的大小。

## 内置变量

`http_v2` 模块支持以下内置变量:

### `$http2`

协商的协议标识符:

|                  |                    |
|------------------|--------------------|
| <code>h2</code>  | 用于 TLS 上的 HTTP/2   |
| <code>h2c</code> | 用于明文 TCP 上的 HTTP/2 |
| <code>""</code>  | 否则为空字符串            |

## HTTP/3

启用客户端连接的 HTTP/3 支持, 以及使用以下 `http_proxy` 指令配置的代理服务器连接:

- `proxy_http3_hq`
- `proxy_http3_max_concurrent_streams`
- `proxy_http3_stream_buffer_size`
- `proxy_http_version`
- `proxy_pass`
- `proxy_quic_active_connection_id_limit`
- `proxy_quic_gso`
- `proxy_quic_host_key`

当从源代码构建时, 此模块默认不构建; 应使用 `--with-http_v3_module` 构建选项启用。

在我们的仓库中的包和镜像中, 构建时已包含该模块。

## 配置示例

```
http {
    log_format quic '$remote_addr - $remote_user [$time_local] '
                   '$request' $status $body_bytes_sent '
                   '$http_referer' '$http_user_agent' '$http3';

    access_log logs/access.log quic;

    server {
        # 为了更好的兼容性, 建议对http/3和https使用相同的端口
        listen 8443 quic reuseport;
        listen 8443 ssl;

        ssl_certificate     certs/example.com.crt;
        ssl_certificate_key certs/example.com.key;

        location / {
            # 用于宣传HTTP/3的可用性
            add_header Alt-Svc 'h3=":8443"; ma=86400';
        }
    }
}
```

### 重要

请注意, 通过 TLS 接受 HTTP/3 连接需要 TLSv1.3 协议支持, 该支持自 OpenSSL 版本 1.1.1 起提供。

## 指令

### http3

|         |                 |
|---------|-----------------|
| Syntax  | http3 on   off; |
| Default | http3 on;       |
| Context | http, server    |

启用 HTTP/3 协议协商。

### http3\_hq

|         |                                 |
|---------|---------------------------------|
| Syntax  | <code>http3_hq on   off;</code> |
| Default | <code>http3_hq off;</code>      |
| Context | <code>http, server</code>       |

启用 QUIC 互操作性测试 中使用的 HTTP/0.9 协议协商。

### http3\_max\_concurrent\_streams

|         |                                                          |
|---------|----------------------------------------------------------|
| Syntax  | <code>http3_max_concurrent_streams <i>number</i>;</code> |
| Default | <code>http3_max_concurrent_streams 128;</code>           |
| Context | <code>http, server</code>                                |

设置连接中并发 HTTP/3 请求流的最大数量。

### http3\_stream\_buffer\_size

|         |                                                    |
|---------|----------------------------------------------------|
| Syntax  | <code>http3_stream_buffer_size <i>size</i>;</code> |
| Default | <code>http3_stream_buffer_size 64k;</code>         |
| Context | <code>http, server</code>                          |

设置用于读取和写入 QUIC 流的缓冲区大小。

### quic\_active\_connection\_id\_limit

|         |                                                             |
|---------|-------------------------------------------------------------|
| Syntax  | <code>quic_active_connection_id_limit <i>number</i>;</code> |
| Default | <code>quic_active_connection_id_limit 2;</code>             |
| Context | <code>http, server</code>                                   |

设置 QUIC `active_connection_id_limit` 传输参数值。这是服务器上可存储的客户端连接 ID 的最大数量。

### quic\_bpf

|         |                                 |
|---------|---------------------------------|
| Syntax  | <code>quic_bpf on   off;</code> |
| Default | <code>quic_bpf off;</code>      |
| Context | main                            |

启用使用 ‘eBPF <<https://ebpf.io/>>’ 的 QUIC 包路由。当启用时, 这允许支持 QUIC 连接迁移。

#### 重要

此指令仅在 Linux 5.7+ 上支持。

### quic\_gso

|         |                                 |
|---------|---------------------------------|
| Syntax  | <code>quic_gso on   off;</code> |
| Default | <code>quic_gso off;</code>      |
| Context | http, server                    |

启用使用分段卸载的优化批量模式发送。

#### 重要

优化发送仅在支持 `UDP_SEGMENT` 的 Linux 上支持。

### quic\_host\_key

|         |                                  |
|---------|----------------------------------|
| Syntax  | <code>quic_host_key file;</code> |
| Default | —                                |
| Context | http, server                     |

设置一个 ‘file’, 用于加密无状态重置和地址验证令牌。默认情况下, 每次重新加载时都会生成一个随机密钥。使用旧密钥生成的令牌将不被接受。



## quic\_retry

|         |                                   |
|---------|-----------------------------------|
| Syntax  | <code>quic_retry on   off;</code> |
| Default | <code>quic_retry off;</code>      |
| Context | http, server                      |

启用 QUIC 地址验证 功能。这包括在 *Retry* 包中发送新令牌或在 *NEW\_TOKEN* 帧中发送新令牌, 以及在 *Initial* 包中验证接收到的令牌。

## 内置变量

*http\_v3* 模块支持以下内置变量:

### `$http3`

协商协议标识符:

|                 |              |
|-----------------|--------------|
| <code>h3</code> | 对于 HTTP/3 连接 |
| <code>hq</code> | 对于 hq 连接     |
| <code>""</code> | 否则为空字符串      |

### `$quic_connection`

QUIC 连接序列号

## XSLT

该模块是一个过滤器, 用于使用一个或多个 XSLT 样式表转换 XML 响应。

当从源代码构建时, 该模块默认不构建; 需要使用 构建选项中的 `--with-http_xslt_module` 来启用。

在我们的代码库中, 该模块是 动态构建的, 并作为一个名为 `angie-module-xslt` 或 `angie-pro-module-xslt` 的独立包提供。

### 🔔 重要

该模块需要 `libxml2` 和 `libxslt` 库。

## 配置示例

```
location / {
    xml_entities    /site/dtd/entities.dtd;
    xslt_stylesheet /site/xslt/one.xslt param=value;
    xslt_stylesheet /site/xslt/two.xslt;
}
```

## 指令

### xml\_entities

|     |                                 |
|-----|---------------------------------|
| 语法  | <code>xml_entities path;</code> |
| 默认  | —                               |
| 上下文 | http, server, location          |

指定声明字符实体的 DTD 文件。该文件在配置阶段被编译。由于技术原因, 模块无法使用在处理的 XML 中声明的外部子集, 因此会被忽略, 而使用一个特别定义的文件。该文件不应描述 XML 结构, 只需声明所需的字符实体即可, 例如:

```
<!ENTITY nbsp "&#xa0;";>
```

### xslt\_last\_modified

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>xslt_last_modified on   off;</code> |
| 默认  | <code>xslt_last_modified off;</code>      |
| 上下文 | http, server, location                    |

允许在 XSLT 转换期间保留原始响应中的“Last-Modified”头字段, 以便于响应缓存。

默认情况下, 由于响应内容在转换过程中被修改并可能包含动态生成的元素或独立于原始响应更改的部分, 因此头字段会被移除。

### xslt\_param

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>xslt_param parameter value;</code> |
| 默认  | —                                        |
| 上下文 | http, server, location                   |

定义 XSLT 样式表的参数。值被视为 XPath 表达式。值可以包含变量。要传递字符串值到样式表, 可以使用 `xslt_string_param` 指令。

可以有多个 `xslt_param` 指令。只有在当前级别没有定义 `xslt_param` 和 `xslt_string_param` 指令时, 这些指令才会从上一级配置继承。

### xslt\_string\_param

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>xslt_string_param parameter value;</code> |
| 默认  | —                                               |
| 上下文 | http, server, location                          |

定义 XSLT 样式表的字符串参数。值中的 XPath 表达式不被解释。值可以包含变量。

可以有多个 `xslt_string_param` 指令。只有在当前级别没有定义 `xslt_param` 和 `xslt_string_param` 指令时, 这些指令才会从上一级配置继承。

### xslt\_stylesheet

|     |                                                                |
|-----|----------------------------------------------------------------|
| 语法  | <code>xslt_stylesheet stylesheet [parameter=value ...];</code> |
| 默认  | —                                                              |
| 上下文 | location                                                       |

定义 XSLT 样式表及其可选参数。样式表在配置阶段被编译。

参数可以单独指定, 也可以使用“:”分隔符组合在一行中。如果参数包含“:”字符, 应将其转义为“%3A”。此外, libxslt 要求将包含非字母数字字符的参数用单引号或双引号括起来, 例如:

```
param1='http%3A//www.example.com':param2=value2
```

参数描述可以包含变量, 例如, 整个参数行可以从单个变量中获取:

```
location / {
    xslt_stylesheet /site/xslt/one.xslt
        $arg_xslt_params
        param1='$value1':param2=value2
```

```
        param3=value3;  
    }
```

可以指定多个样式表。它们将按指定顺序依次应用。

### xslt\_types

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>xslt_types mime-type ...;</code> |
| 默认  | <code>xslt_types text/xml;</code>      |
| 上下文 | http, server, location                 |

在响应中除“*text/xml*”之外，启用指定 MIME 类型的转换。特殊值“\*”匹配任何 MIME 类型。如果转换结果是 HTML 响应，其 MIME 类型将更改为“*text/html*”。

核心 HTTP 模块实现了 HTTP 服务器的基本功能，包括定义服务器块、配置用于请求路由的位置、提供静态文件和控制访问、配置重定向、支持 *keep-alive* 连接以及管理请求和响应头。

本节中的其他模块扩展了此功能，使您能够灵活配置和优化 HTTP 服务器，以满足各种场景和需求。

## 指令

### absolute\_redirect

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>absolute_redirect on   off;</code> |
| 默认值 | <code>absolute_redirect on;</code>       |
| 上下文 | http, server, location                   |

如果禁用，Angie 发出的重定向将是相对的。

另请参见 `server_name_in_redirect` 和 `port_in_redirect` 指令。

### aio

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>aio on   off   threads` [= `pool];</code> |
| 默认值 | <code>aio off;</code>                           |
| 上下文 | http, server, location                          |

启用或禁用在 FreeBSD 和 Linux 上使用异步文件 I/O (AIO):

```
location /video/ {
    aio          on;
    output_buffers 1 64k;
}
```

在 FreeBSD 上, 从 FreeBSD 4.3 开始可以使用 AIO。在 FreeBSD 11.0 之前, AIO 可以静态链接到内核中:

```
options VFS_AIO
```

或作为可加载内核模块动态加载:

```
kldload aio
```

在 Linux 上, 从内核版本 2.6.22 开始可以使用 AIO。此外, 有必要启用 *directio*, 否则读取将是阻塞的:

```
location /video/ {
    aio          on;
    directio     512;
    output_buffers 1 128k;
}
```

在 Linux 上, *directio* 只能用于读取 512 字节边界 (或 XFS 的 4K) 对齐的块。文件未对齐的结尾以阻塞模式读取。对文件开头和结尾未对齐数据的字节范围请求和 FLV 请求也是如此: 文件开头和结尾的未对齐数据读取将是阻塞的。

当在 Linux 上同时启用 AIO 和 *sendfile* 时, AIO 用于大于或等于 *directio* 指令中指定大小的文件, 而 *sendfile* 用于较小尺寸的文件或禁用 *directio* 时:

```
location /video/ {
    sendfile     on;
    aio          on;
    directio     8m;
}
```

最后, 可以通过多线程读取和发送文件, 而不阻塞工作进程:

```
location /video/ {
    sendfile     on;
    aio          threads;
}
```

读取和发送文件操作被卸载到指定 *pool* 的线程中。如果省略了 *pool* 名称, 则使用名称为 “default” 的池。池名称也可以用变量设置:

```
aio threads=pool$disk;
```

默认情况下, 多线程是禁用的, 需要使用 `--with-threads` 配置参数启用。目前, 多线程仅与 *epoll*、*kqueue*

和 `eventport` 方法兼容。多线程文件发送仅在 Linux 上受支持。

另请参见 `sendfile` 指令。

### `aio_write`

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>aio_write on   off;</code>    |
| 默认值 | <code>aio_write off;</code>         |
| 上下文 | <code>http, server, location</code> |

如果启用了 `aio`, 请指定是否用于写入文件。目前, 这仅在使用 `aio threads` 时有效, 并且仅限于写入从代理服务器接收到的数据的临时文件。

### `alias`

|     |                          |
|-----|--------------------------|
| 语法  | <code>alias path;</code> |
| 默认值 | —                        |
| 上下文 | <code>location</code>    |

定义指定位置的替代。例如, 使用以下配置:

```
location /i/ {
    alias /data/w3/images/;
}
```

在请求 `/i/top.gif` 时, 将发送文件 `/data/w3/images/top.gif`。

`path` 值可以包含变量, 除了 `$document_root` 和 `$realpath_root`。

如果在使用正则表达式定义的 `location` 中使用 `alias`, 则此类正则表达式应包含捕获, 并且 `alias` 应引用这些捕获, 例如:

```
location ~ ^/users/(.+\.(?:gif|jpe?g|png))$ {
    alias /data/w3/images/$1;
}
```

当位置匹配指令值的最后部分时:

```
location /images/ {
    alias /data/w3/images/;
}
```

最好使用 `root` 指令代替:

```
location /images/ {  
    root /data/w3;  
}
```

### auth\_delay

|     |                               |
|-----|-------------------------------|
| 语法  | <code>auth_delay time;</code> |
| 默认值 | <code>auth_delay 0s;</code>   |
| 上下文 | http, server, location        |

延迟处理带有 401 响应代码的未经授权请求, 以防止在访问受 *password* 或子请求结果 限制时进行时间攻击。

### auto\_redirect

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>auto_redirect [on   off   default];</code> |
| 默认值 | <code>auto_redirect default;</code>              |
| 上下文 | http, server, location                           |

该指令控制前缀位置以斜杠结束时的重定向 行为:

```
location /prefix/ {  
    auto_redirect on;  
}
```

在此, 请求 `/prefix` 会导致重定向到 `/prefix/`。

将此设置为 `on` 和 `off` 可以显式启用和禁用重定向。设置为 `default` 时, 只有在 `location` 处理带有 *api*、*proxy\_pass*、*fastcgi\_pass*、*uwsgi\_pass*、*scgi\_pass*、*memcached\_pass* 或 *grpc\_pass* 的请求时, 才启用重定向。

### chunked\_transfer\_encoding

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>chunked_transfer_encoding on   off;</code> |
| 默认值 | <code>chunked_transfer_encoding on;</code>       |
| 上下文 | http, server, location                           |

允许在 HTTP/1.1 中禁用分块传输编码。当使用不支持分块编码的软件时, 这可能会派上用场, 尽管标准要求支持。

### client\_body\_buffer\_size

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>client_body_buffer_size size;</code>   |
| 默认值 | <code>client_body_buffer_size 8k 16k;</code> |
| 上下文 | http, server, location                       |

设置用于读取客户端请求体的缓冲区大小。如果请求体大于缓冲区，则整个请求体或其一部分将写入临时文件。默认情况下，缓冲区大小等于两个内存页。在 x86 和其他 32 位平台以及 x86-64 上为 8K。在其他 64 位平台上通常为 16K。

### client\_body\_in\_file\_only

|     |                                                         |
|-----|---------------------------------------------------------|
| 语法  | <code>client_body_in_file_only on   clean   off;</code> |
| 默认值 | <code>client_body_in_file_only off;</code>              |
| 上下文 | http, server, location                                  |

确定 `Angie` 是否应将整个客户端请求体保存到文件中。此指令可在调试期间使用，或在使用 `$request_body_file` 变量或模块 `Perl` 的 `$r->request_body_file` 方法时使用。

当设置为 `on` 时，请求处理后不会删除临时文件。

|                    |                    |
|--------------------|--------------------|
| <code>on</code>    | 请求处理后不会删除临时文件      |
| <code>clean</code> | 将导致请求处理后留下的临时文件被删除 |

### client\_body\_in\_single\_buffer

|     |                                                     |
|-----|-----------------------------------------------------|
| 语法  | <code>client_body_in_single_buffer on   off;</code> |
| 默认值 | <code>client_body_in_single_buffer off;</code>      |
| 上下文 | http, server, location                              |

确定 `Angie` 是否应将整个客户端请求体保存在单个缓冲区中。推荐在使用 `$request_body` 变量时使用此指令，以减少涉及的复制操作数量。



### client\_body\_temp\_path

|     |                                                                                                       |
|-----|-------------------------------------------------------------------------------------------------------|
| 语法  | <code>client_body_temp_path path [level1 [level2 [level3]]];</code>                                   |
| 默认值 | <code>client_body_temp_path client_body_temp;</code> (路径取决于 <code>--http-proxy-temp-path</code> 构建选项) |
| 上下文 | http, server, location                                                                                |

定义用于存储持有客户端请求体的临时文件的目录。可以在指定目录下使用最多三级子目录层次结构。例如, 在以下配置中

```
client_body_temp_path /spool/angie/client_temp 1 2;
```

临时文件的路径可能如下所示:

```
/spool/angie/client_temp/7/45/00000123457
```

### client\_body\_timeout

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>client_body_timeout time;</code> |
| 默认值 | <code>client_body_timeout 60s;</code>  |
| 上下文 | http, server, location                 |

定义读取客户端请求体的超时时间。超时时间仅设置为两次连续读操作之间的时间段, 而不是整个请求体的传输时间。如果客户端在此时间内未传输任何内容, 则请求将以 408 (请求超时) 错误终止。

### client\_header\_buffer\_size

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>client_header_buffer_size size;</code> |
| 默认值 | <code>client_header_buffer_size 1k;</code>   |
| 上下文 | http, server                                 |

设置用于读取客户端请求头的缓冲区大小。对于大多数请求, 1K 字节的缓冲区就足够了。然而, 如果请求包含很长的 cookies, 或者来自 WAP 客户端, 可能无法适应 1K 的缓冲区。如果请求行或请求头字段不适合这个缓冲区, 将分配由 `large_client_header_buffers` 指令配置的更大缓冲区。

如果在 `server` 级别指定了该指令, 则可以使用默认服务器的值。详情请参见 [虚拟服务器选择](#) 部分。

### client\_header\_timeout

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>client_header_timeout time;</code> |
| 默认值 | <code>client_header_timeout 60s;</code>  |
| 上下文 | http, server                             |

定义读取客户端请求头的超时时间。如果客户端在此时间内未传输完整的请求头, 则请求终止并返回 408 (请求超时) 错误。

### client\_max\_body\_size

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>client_max_body_size size;</code> |
| 默认值 | <code>client_max_body_size 1m;</code>   |
| 上下文 | http, server, location                  |

设置客户端请求体的最大允许大小。如果请求的大小超过配置的值, 将返回 413 (请求实体过大) 错误给客户端。请注意, 浏览器无法正确显示此错误。

|   |                |
|---|----------------|
| 0 | 禁用对客户端请求体大小的检查 |
|---|----------------|

### connection\_pool\_size

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>connection_pool_size size;</code>      |
| 默认值 | <code>connection_pool_size 256   512;</code> |
| 上下文 | http, server, location                       |

允许精确调整每个连接的内存分配。此指令对性能影响最小, 通常不应使用。

默认值:

|          |        |
|----------|--------|
| 256 (字节) | 32 位平台 |
| 512 (字节) | 64 位平台 |

## default\_type

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>default_type mime-type;</code>  |
| 默认值 | <code>default_type text/plain;</code> |
| 上下文 | http, server, location                |

定义响应的默认 MIME 类型。文件名扩展名到 MIME 类型的映射可以通过 *types* 指令设置。

## directio

|     |                                   |
|-----|-----------------------------------|
| 语法  | <code>directio size   off;</code> |
| 默认值 | <code>directio off;</code>        |
| 上下文 | http, server, location            |

在读取大于或等于指定大小的文件时, 启用使用 *O\_DIRECT* 标志 (FreeBSD, Linux)、*F\_NOCACHE* 标志 (macOS) 或 *directio()* 函数 (Solaris)。该指令会自动禁用给定请求的 *sendfile* 使用。它对于服务大文件很有用:

```
directio 4m;
```

或者在 Linux 上使用 *aio* 时。

## directio\_alignment

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>directio_alignment size;</code> |
| 默认值 | <code>directio_alignment 512;</code>  |
| 上下文 | http, server, location                |

设置 *directio* 的对齐。在大多数情况下, 512 字节的对齐就足够了。然而, 在 Linux 下使用 XFS 时, 需要增加到 4K。

## disable\_symlinks

|     |                                                                                                    |
|-----|----------------------------------------------------------------------------------------------------|
| 语法  | <code>disable_symlinks off;</code><br><code>disable_symlinks on   if_not_owner [from=part];</code> |
| 默认值 | <code>disable_symlinks off;</code>                                                                 |
| 上下文 | http, server, location                                                                             |

确定打开文件时符号链接的处理方式:

|                           |                                                                                                                                                                                                                                         |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>off</code>          | 路径名中的符号链接是允许的且不检查。这是默认行为。                                                                                                                                                                                                               |
| <code>on</code>           | 如果路径名的任何组件是符号链接, 则拒绝对文件的访问。                                                                                                                                                                                                             |
| <code>if_not_owner</code> | 如果路径名的任何组件是符号链接, 并且链接和链接指向的对象有不同的所有者, 则拒绝对文件的访问。                                                                                                                                                                                        |
| <code>from=part</code>    | 在检查符号链接时 (参数 <code>on</code> 和 <code>if_not_owner</code> ), 通常会检查路径名的所有组件。通过另外指定 <code>from=part</code> 参数, 可以避免检查路径名的初始部分的符号链接。在这种情况下, 仅检查从指定初始部分之后的路径名组件的符号链接。如果值不是检查的路径名的初始部分, 则将整个路径名视为未指定该参数时进行检查。如果值与整个文件名匹配, 则不检查符号链接。参数值可以包含变量。 |

示例:

```
disable_symlinks on from=$document_root;
```

此指令仅在具有 `openat()` 和 `fstatat()` 接口的系统上可用。此类系统包括现代版本的 FreeBSD、Linux 和 Solaris。

#### 警告

参数 `on` 和 `if_not_owner` 会增加处理开销。

在不支持仅用于搜索打开目录的系统上, 使用这些参数需要工作进程对所有被检查的目录具有读取权限。

#### 备注

自动索引、随机索引 和 *DAV* 模块当前忽略此指令。

## error\_page

|     |                                                      |
|-----|------------------------------------------------------|
| 语法  | <code>error_page code ... [= [response]] uri;</code> |
| 默认值 | —                                                    |
| 上下文 | http, server, location, if in location               |

定义将在指定错误时显示的 URI *uri* 值可以包含变量。

示例:

```
error_page 404 /404.html;
error_page 500 502 503 504 /50x.html;
```

这会将请求方法更改为“GET” (除“GET” 和“HEAD” 以外的所有方法), 并内部重定向到指定的 *uri*。

此外, 可以使用 ‘=response’语法更改响应代码, 例如:

```
error_page 404 =200 /empty.gif;
```

如果错误响应由代理服务器或 FastCGI/uwsgi/SCGI/gRPC 服务器处理, 并且服务器可能返回不同的响应代码 (例如 200、302、401 或 404), 可以响应其返回的代码:

```
error_page 404 = /404.php;
```

如果在内部重定向期间不需要更改 URI 和方法, 可以将错误处理传递到命名位置:

```
location / {  
    error_page 404 = @fallback;  
}  
  
location @fallback {  
    proxy_pass http://backend;  
}
```

#### 备注

如果 uri 处理导致错误, 最后发生的错误的状态代码将返回给客户端。

还可以使用 URL 重定向进行错误处理:

```
error_page 403      http://example.com/forbidden.html;  
error_page 404 =301 http://example.com/notfound.html;
```

在这种情况下, 默认情况下, 响应代码 302 返回给客户端。它只能更改为重定向状态代码之一 (301、302、303、307 和 308)。

## etag

|     |                        |
|-----|------------------------|
| 语法  | etag on   off;         |
| 默认值 | etag on;               |
| 上下文 | http, server, location |

启用或禁用静态资源 “ETag” 响应头字段的自动生成。

## http

|     |                           |
|-----|---------------------------|
| 语法  | <code>http { ... }</code> |
| 默认值 | —                         |
| 上下文 | main                      |

提供指定 HTTP 服务器指令的配置文件上下文。

## if\_modified\_since

|     |                                                      |
|-----|------------------------------------------------------|
| 语法  | <code>if_modified_since off   exact   before;</code> |
| 默认值 | <code>if_modified_since exact;</code>                |
| 上下文 | http, server, location                               |

指定如何将响应的修改时间与 ‘If-Modified-Since’ 请求头字段中的时间进行比较：

|        |                                             |
|--------|---------------------------------------------|
| off    | 响应始终被视为已修改                                  |
| exact  | 精确匹配                                        |
| before | 响应的修改时间小于或等于 ‘If-Modified-Since’ 请求头字段中的时间。 |

## ignore\_invalid\_headers

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>ignore_invalid_headers on   off;</code> |
| 默认值 | <code>ignore_invalid_headers on;</code>       |
| 上下文 | http, server                                  |

控制是否应忽略具有无效名称的头字段。有效名称由英文字母、数字、连字符和可能的下划线组成 (由 `underscores_in_headers` 指令控制)。

如果在 `server` 级别指定了该指令, 则可以使用默认服务器的值。

## internal

|     |                        |
|-----|------------------------|
| 语法  | <code>internal;</code> |
| 默认值 | —                      |
| 上下文 | location               |

指定给定位置只能用于内部请求。对于外部请求, 返回客户端错误 404 (未找到)。内部请求包括以下内容:

- 由 `error_page`、`index`、`random_index` 和 `try_files` 指令重定向的请求;
- 由上游服务器的 `X-Accel-Redirect` 响应头字段重定向的请求;
- 由 `http_ssi` 模块的 `'include virtual'` 命令、`http_addition` 模块指令以及 `auth_request` 和 `mirror` 指令形成的子请求;
- 由 `rewrite` 指令更改的请求。

示例:

```
error_page 404 /404.html;  
  
location = /404.html {  
    internal;  
}
```

#### 备注

每个请求的内部重定向限制为 10 次, 以防止在不正确配置中可能发生的请求处理循环。如果达到此限制, 则返回错误 500 (内部服务器错误)。在这种情况下, 错误日志中可能会看到 `rewrite` 或内部重定向循环消息。

## keepalive\_disable

|     |                                                                 |
|-----|-----------------------------------------------------------------|
| 语法  | <code>keepalive_disable none   browser ...;</code>              |
| 默认值 | <code>keepalive_disable msie6;</code>                           |
| 上下文 | <code>http</code> , <code>server</code> , <code>location</code> |

禁用与行为不当的浏览器的 keep-alive 连接。 `browser` 参数指定将影响哪些浏览器。

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <code>none</code>   | 允许与所有浏览器的 keep-alive 连接                          |
| <code>msie6</code>  | 在接收到 POST 请求后, 禁用与旧版 MSIE 的 keep-alive 连接        |
| <code>safari</code> | 禁用与 macOS 及类似操作系统上的 Safari 及类似浏览器的 keep-alive 连接 |

### keepalive\_requests

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>keepalive_requests number;</code> |
| 默认值 | <code>keepalive_requests 1000;</code>   |
| 上下文 | http, server, location                  |

设置可以通过一个 keep-alive 连接服务的最大请求数。在达到最大请求数后, 连接将关闭。

定期关闭连接是必要的, 以便释放每个连接的内存分配。因此, 使用过高的最大请求数可能导致内存使用过多, 不推荐这样做。

### keepalive\_time

|     |                                   |
|-----|-----------------------------------|
| 语法  | <code>keepalive_time time;</code> |
| 默认值 | <code>keepalive_time 1h;</code>   |
| 上下文 | http, server, location            |

限制通过一个 keep-alive 连接处理请求的最长时间。达到此时间后, 在后续请求处理后关闭连接。

### keepalive\_timeout

|     |                                                          |
|-----|----------------------------------------------------------|
| 语法  | <code>keepalive_timeout timeout [header_timeout];</code> |
| 默认值 | <code>keepalive_timeout 75s;</code>                      |
| 上下文 | http, server, location                                   |

|         |                                             |
|---------|---------------------------------------------|
| timeout | 设置一个超时时间, 在此期间 keep-alive 客户端连接将在服务器端保持打开状态 |
| 0       | 禁用 keep-alive 客户端连接                         |

可选的第二个参数在 “Keep-Alive: timeout=time” 响应头字段中设置一个值。两个参数可以不同。

“Keep-Alive: timeout=time” 头字段被 Mozilla 和 Konqueror 识别。MSIE 会在大约 60 秒内自行关闭 keep-alive 连接。



## large\_client\_header\_buffers

|     |                                                       |
|-----|-------------------------------------------------------|
| 语法  | <code>large_client_header_buffers number size;</code> |
| 默认值 | <code>large_client_header_buffers 4 8k;</code>        |
| 上下文 | http, server                                          |

设置用于读取大客户端请求头的缓冲区的最大数量和大小。请求行不能超过一个缓冲区的大小，否则将返回 414（请求 URI 太大）错误给客户端。请求头字段也不能超过一个缓冲区的大小，否则将返回 400（错误请求）错误给客户端。缓冲区仅在需要时分配。默认情况下，缓冲区大小等于 8K 字节。如果在请求处理结束后连接转入 keep-alive 状态，这些缓冲区将被释放。

如果指令在 *server* 级别指定，则可以使用默认服务器的值。

## limit\_except

|     |                                                         |
|-----|---------------------------------------------------------|
| 语法  | <code>limit_except method1 [method2...] { ... };</code> |
| 默认值 | —                                                       |
| 上下文 | location                                                |

限制在某个位置内允许的 HTTP 方法。*method* 可以是以下之一：GET, HEAD, POST, PUT, DELETE, MKCOL, COPY, MOVE, OPTIONS, PROPFIND, PROPPATCH, LOCK, UNLOCK 或 PATCH。允许 GET 方法也启用 HEAD 方法。可以使用访问控制 和基础认证 模块中的指令限制对其他方法的访问：

```
limit_except GET {  
    allow 192.168.1.0/32;  
    deny all;  
}
```

### 备注

此示例将限制对所有方法的访问，除了 GET 和 HEAD。

## limit\_rate

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>limit_rate rate;</code>          |
| 默认值 | <code>limit_rate 0;</code>             |
| 上下文 | http, server, location, if in location |

限制向客户端传输响应的速率。速率以每秒字节数指定。零值禁用速率限制。限制是针对每个请求设置的，因此如果客户端同时打开两个连接，总速率将是指定限制的两倍。

参数值可以包含变量。在某些情况下需要根据特定条件限制速率时可能有用：

```
map $slow $rate {  
    1     4k;  
    2     8k;  
}  
  
limit_rate $rate;
```

也可以在 `$limit_rate` 变量中设置速率限制，但不推荐使用此方法：

```
server {  
  
    if ($slow) {  
        set $limit_rate 4k;  
    }  
  
}
```

速率限制也可以在代理服务器响应的“X-Accel-Limit-Rate”头字段中设置。可以使用 `proxy_ignore_headers`, `fastcgi_ignore_headers`, `uwsgi_ignore_headers` 和 `scgi_ignore_headers` 指令禁用此功能。

### limit\_rate\_after

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>limit_rate_after size;</code>    |
| 默认值 | <code>limit_rate_after 0;</code>       |
| 上下文 | http, server, location, if in location |

设置初始量，超过此量后将限制向客户端传输响应的速率。参数值可以包含变量。

示例：

```
location /flv/ {  
    flv;  
    limit_rate_after 500k;  
    limit_rate      50k;  
}
```

## lingering\_close

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>lingering_close on   always   off;</code> |
| 默认值 | <code>lingering_close on;</code>                |
| 上下文 | http, server, location                          |

控制 Angie 如何关闭客户端连接。

|        |                                                                |
|--------|----------------------------------------------------------------|
| on     | 指示 Angie 等待并处理来自客户端的额外数据, 然后再完全关闭连接, 但仅在启发式方法建议客户端可能正在发送更多数据时。 |
| always | 将导致 Angie 无条件地等待和处理客户端的额外数据。                                   |
| off    | 告诉 Angie 不要等待更多数据, 并立即关闭连接。这种行为会破坏协议, 不应在正常情况下使用。              |

要控制关闭 HTTP/2 连接, 必须在 *server* 级别指定指令。

## lingering\_time

|     |                                   |
|-----|-----------------------------------|
| 语法  | <code>lingering_time time;</code> |
| 默认值 | <code>lingering_time 30s;</code>  |
| 上下文 | http, server, location            |

当 *lingering\_close* 生效时, 此指令指定 Angie 将处理 (读取并忽略) 来自客户端的额外数据的最长时间。之后, 即使有更多数据, 连接也将关闭。

## lingering\_timeout

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>lingering_timeout time;</code> |
| 默认值 | <code>lingering_timeout 5s;</code>   |
| 上下文 | http, server, location               |

当 *lingering\_close* 生效时, 此指令指定等待更多客户端数据到达的最长时间。如果在此时间内没有收到数据, 连接将关闭。否则, 数据将被读取并忽略, Angie 将开始等待更多数据。该“等待-读取-忽略”循环会重复, 但不会超过 *lingering\_time* 指令指定的时间。

## listen

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <pre>listen address[:port] [default_server] [ssl] [http2   quic] [proxy_protocol] [setfib=number] [fastopen=number] [backlog=number] [rcvbuf=size] [sndbuf=size] [accept_filter=filter] [deferred] [bind] [ipv6only=on   off] [reuseport] [so_keepalive=on off][keepidle]:[keepintvl]:[keepcnt]]; listen port [default_server] [ssl] [http2   quic] [proxy_protocol] [setfib=number] [fastopen=number] [backlog=number] [rcvbuf=size] [sndbuf=size] [accept_filter=filter] [deferred] [bind] [ipv6only=on   off] [reuseport] [so_keepalive=on off][keepidle]:[keepintvl]:[keepcnt]]; listen unix:path [default_server] [ssl] [http2   quic] [proxy_protocol] [backlog=number] [rcvbuf=size] [sndbuf=size] [accept_filter=filter] [deferred] [bind] [so_keepalive=on off][keepidle]:[keepintvl]:[keepcnt];</pre> |
| 默认值 | <pre>listen *:80   *:8000;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| 上下文 | <pre>server</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

设置用于监听套接字的 *address* 和 *port*, 或服务器将接受请求的 UNIX 域套接字的路径。可以同时指定 *address* 和 *port*, 或者只指定 *address* 或只指定 *port*。 *address* 也可以是主机名, 例如:

```
listen 127.0.0.1:8000;
listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000;
```

IPv6 地址在方括号中指定:

```
listen [::]:8000;
listen [::1];
```

UNIX 域套接字使用 `unix:` 前缀指定:

```
listen unix:/var/run/angie.sock;
```

如果仅给出 ‘address’, 则使用端口 ‘80’。

如果指令不存在, 则如果 Angie 以超用户权限运行, 则使用 ‘\*:80’, 否则使用 ‘\*:8000’。

|                             |                                                                                                                                               |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>default_server</code> | 指定此参数的服务器将成为给定 <code>address:port</code> 对的默认服务器（两者共同形成一个 监听套接字）。<br>如果没有带有 <code>default_server</code> 参数的指令，该监听套接字的默认服务器将是配置中第一个服务此套接字的服务器。 |
| <code>ssl</code>            | 允许指定在此端口接受的所有连接都应以 SSL 模式工作。这使得处理 HTTP 和 HTTPS 请求的服务器配置更为 紧凑。                                                                                 |
| <code>http2</code>          | 配置端口以接受 HTTP/2 连接。通常，为了使其工作还应指定 <code>ssl</code> 参数，但 Angie 也可以配置为在没有 SSL 的情况下接受 HTTP/2 连接。<br>自 1.2.0 版本弃用。<br>请使用 <code>http2</code> 指令。    |
| <code>quic</code>           | 配置端口以接受 QUIC 连接。要使用此选项，必须启用并配置 Angie 的 <code>HTTP3</code> 模块。设置 <code>quic</code> 后，还可以指定 <code>reuseport</code> 以便使用多个工作进程。                  |
| <code>proxy_protocol</code> | 允许指定在此端口接受的所有连接应使用 PROXY 协议。                                                                                                                  |

`listen` 指令可以有几个特定于套接字相关系统调用的附加参数。这些参数可以在任何 `listen` 指令中指定，但对于给定的 `address:port` 对只能指定一次。

|                              |                                                                     |
|------------------------------|---------------------------------------------------------------------|
| <code>setfib=number</code>   | 此参数为监听套接字设置相关的路由表，FIB（ <code>SO_SETFIB</code> 选项）。目前仅在 FreeBSD 上有效。 |
| <code>fastopen=number</code> | 为监听套接字启用“TCP Fast Open”并限制尚未完成三次握手的连接队列的最大长度。                       |

#### 小心

除非服务器能够处理多次接收相同的数据 SYN 包，否则不要启用此功能。

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>backlog=number</code>       | 设置 <code>listen()</code> 调用中的 <code>backlog</code> 参数, 限制待决连接队列的最大长度。默认情况下, FreeBSD、DragonFly BSD 和 macOS 上的 <code>backlog</code> 设置为 -1, 其他平台上设置为 511。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <code>rcvbuf=size</code>          | 为监听套接字设置接收缓冲区大小 ( <code>SO_RCVBUF</code> 选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>sndbuf=size</code>          | 为监听套接字设置发送缓冲区大小 ( <code>SO_SNDBUF</code> 选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>accept_filter=filter</code> | 设置监听套接字的接受过滤器名称 ( <code>SO_ACCEPTFILTER</code> 选项), 在传递给 <code>accept()</code> 之前过滤传入连接。这仅在 FreeBSD 和 NetBSD 5.0+ 上有效。可能的值为 <code>dataready</code> 和 <code>httpready</code> 。                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>deferred</code>             | 指示在 Linux 上使用延迟的 <code>accept()</code> ( <code>TCP_DEFER_ACCEPT</code> 套接字选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <code>bind</code>                 | 指示为给定的 <code>address:port</code> 对进行单独的 <code>bind()</code> 调用。这很有用, 因为如果有多个 <code>listen</code> 指令使用相同的端口但不同的地址, 并且其中一个 <code>listen</code> 指令监听该给定 <code>port</code> 的所有地址 ( <code>*:port</code> ), <code>Angie</code> 将仅绑定到 <code>*:port</code> 。应该注意的是, 在这种情况下将进行 <code>getsockname()</code> 系统调用以确定接受连接的地址。如果使用了 <code>setfib</code> 、 <code>fastopen</code> 、 <code>backlog</code> 、 <code>rcvbuf</code> 、 <code>sndbuf</code> 、 <code>accept_filter</code> 、 <code>deferred</code> 、 <code>ipv6only</code> 、 <code>reuseport</code> 或 <code>so_keepalive</code> 参数, 则总是会为给定的 <code>address:port</code> 对进行单独的 <code>bind()</code> 调用。 |
| <code>ipv6only=on   off</code>    | 此参数通过 <code>IPV6_V6ONLY</code> 套接字选项决定是否仅接受 IPv6 连接或同时接受 IPv6 和 IPv4 连接。默认情况下此参数为开启状态。启动时只能设置一次。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>reuseport</code>            | 此参数指示为每个工作进程创建一个单独的监听套接字 (在 Linux 3.9+ 和 DragonFly BSD 上使用 <code>SO_REUSEPORT</code> 套接字选项, 或在 FreeBSD 12+ 上使用 <code>SO_REUSEPORT_LB</code> ), 允许内核在工作进程之间分配传入连接。目前仅在 Linux 3.9+、DragonFly BSD 和 FreeBSD 12+ 上有效。                                                                                                                                                                                                                                                                                                                                                                                                                            |

### 小心

不当使用此选项可能会带来安全隐患。

`so_keepalive=on | off | [keepidle]:[keepintvl]:[keepcnt]`

配置监听套接字的“TCP keepalive”行为。

|                  |                                     |
|------------------|-------------------------------------|
| <code>''</code>  | 如果省略此参数, 则套接字将默认使用操作系统的设置           |
| <code>on</code>  | 为套接字开启 <code>SO_KEEPALIVE</code> 选项 |
| <code>off</code> | 为套接字关闭 <code>SO_KEEPALIVE</code> 选项 |

某些操作系统支持使用 `TCP_KEEPIDLE`、`TCP_KEEPINTVL` 和 `TCP_KEEPCNT` 套接字选项按每个套接字设置 TCP keepalive 参数。在这些系统上 (目前为 Linux 2.4+、NetBSD 5+ 和 FreeBSD 9.0-STABLE), 可以使用 `keepidle`、`keepintvl` 和 `keepcnt` 参数进行配置。可以省略一个或两个参数, 在这种情况下, 将使用对应套接字选项的系统默认设置。例如,

```
so_keepalive=30m::10
```

将空闲超时 (`TCP_KEEPIDLE`) 设置为 30 分钟, 探测间隔 (`TCP_KEEPINTVL`) 保持系统默认, 探

测次数 (`TCP_KEEPCNT`) 设置为 10 次。

示例:

```
listen 127.0.0.1 default_server accept_filter=dataready backlog=1024;
```

## location

|     |                                                                  |
|-----|------------------------------------------------------------------|
| 语法  | <code>location ([ =   ~   ~*   ^~ ] uri   @name)+ { ... }</code> |
| 默认值 | —                                                                |
| 上下文 | server, location                                                 |

设置配置, 具体取决于请求 URI 是否匹配任何匹配表达式。

匹配是针对规范化的 URI 进行的, 解码“%XX”形式编码的文本, 解析对相对路径组件“?”和“..”的引用, 以及可能的压缩 两个或多个相邻斜杠为单个斜杠。

`location` 可以由前缀字符串或正则表达式定义。

正则表达式是通过前缀修饰符指定的:

|                 |          |
|-----------------|----------|
| <code>~*</code> | 不区分大小写匹配 |
| <code>~</code>  | 区分大小写匹配  |

为了找到匹配请求的 `location`, Angie 首先检查用前缀字符串定义的 `locations` (称为前缀 `locations`)。在它们中, 选择匹配前缀最长的 `location` 并暂时存储。

### 备注

对于像 macOS 这样的不区分大小写的操作系统, 前缀字符串匹配是不区分大小写的。然而, 匹配仅限于单字节区域。

然后, 按它们在配置文件中的出现顺序评估基于正则表达式的 `locations`。它们的评估在第一次匹配时停止, 并使用相应的配置。如果未找到匹配的正则表达式 `location`, Angie 使用暂时存储的前缀 `location` 的配置。

除下述例外, `location` 块可以嵌套。

正则表达式 `locations` 可以定义捕获组, 稍后可以与其他指令一起使用。

如果匹配的前缀 `location` 使用了 `^~` 修饰符, 则不检查正则表达式 `locations`。

此外, `=` 修饰符为 `location` 启用精确 URI 匹配模式; 如果找到精确匹配, 则查找停止。例如, 如果 `/` 请求频繁, 则定义 `location =/` 会加速其处理, 因为查找在精确匹配处停止。显然, 这样的 `locations` 不能包含嵌套的 `locations`。

示例:

```
location =/ {
    #配置 A
}

location / {
    #配置 B
}

location /documents/ {
    #配置 C
}

location ^~/images/ {
    #配置 D
}

location ~*\.(gif|jpg|jpeg)$ {
    #配置 E
}
```

- 一个 / 请求匹配配置 A,
- 一个 /index.html 请求匹配配置 B,
- 一个 /documents/document.html 请求匹配配置 C,
- 一个 /images/1.gif 请求匹配配置 D,
- 一个 /documents/1.jpg 请求匹配配置 E。

#### **i** 备注

如果一个前缀 location 以斜杠字符结尾并且启用了 *auto\_redirect*, 则会发生以下情况: 当一个请求到达时, 其 URI 没有尾部斜杠但其他方面完全匹配前缀, 将返回一个永久 301 代码重定向, 指向带有附加斜杠的请求 URI。

对于精确 URI 匹配的 location, 不适用重定向:

```
location /user/ {
    proxy_pass http://user.example.com;
}

location =/user {
    proxy_pass http://login.example.com;
}
```

@ 前缀定义了一个命名 *location*。这些 locations 不用于常规请求处理, 但可以用于请求重定向。它们不能嵌套, 也不能包含嵌套的 locations。



## 组合 locations

几个定义了相同配置块的 *location* 上下文可以通过在一个 *location* 中列出其所有匹配表达式并包含一个配置块来压缩。这被称为 组合 *location*。

假设前面示例中的配置 A、D 和 E 定义了相同的配置；你可以将它们合并为一个 *location*：

```
location =/  
    ^~/images/  
    ~*\.(gif|jpg|jpeg)$ {  
    # 通用配置  
}
```

一个命名的 *location* 也可以是组合的一部分：

```
location =/  
    @named_combined {  
    #...  
}
```

### 小心

一个组合 *location* 在匹配表达式和其修饰符之间不能有空格。正确形式：`location ~*/match(ing|es|er)$ ...`。

### 备注

目前，一个组合 *location* 不能立即包含带有 URI 设置的 *proxy\_pass* 和类似指令，也不能包含 *api* 或 *alias*。然而，这些指令可以由嵌套在组合 *location* 中的 *locations* 使用。

## log\_not\_found

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>log_not_found on   off;</code> |
| 默认值 | <code>log_not_found on;</code>       |
| 上下文 | <code>http, server, location</code>  |

启用或禁用将未找到文件的错误记录到 *error\_log*。

### log\_subrequest

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>log_subrequest on   off;</code> |
| 默认值 | <code>log_subrequest off;</code>      |
| 上下文 | http, server, location                |

启用或禁用将子请求记录到`access_log`。

### max\_ranges

|     |                                |
|-----|--------------------------------|
| 语法  | <code>max_ranges number</code> |
| 默认值 | —                              |
| 上下文 | http, server, location         |

限制字节范围请求中允许的最大范围数。超过限制的请求将被视为没有指定字节范围来处理。默认情况下, 范围数是没有限制的。

|   |            |
|---|------------|
| 0 | 完全禁用字节范围支持 |
|---|------------|

### merge\_slashes

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>merge_slashes on   off;</code> |
| 默认值 | <code>merge_slashes on;</code>       |
| 上下文 | http, server                         |

启用或禁用将 URI 中两个或多个相邻斜杠除压缩成单个斜杠。

请注意, 压缩对于前缀字符串和正则表达式位置的正确匹配是至关重要的。没有它, `//scripts/one.php` 请求将不会匹配

```
location /scripts/ { }
```

并可能被处理为静态文件。因此它会被转换为 `/scripts/one.php`。

在 URI 包含 base64 编码名称的情况下, 关闭压缩可能是必要的, 因为 `base64` 内部使用了 `"/` 字符。然而, 出于安全考虑, 最好避免关闭压缩。

如果指令是在 `server` 级别指定的, 则可以使用默认服务器中的值。

### msie\_padding

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>msie_padding on   off;</code> |
| 默认值 | <code>msie_padding on;</code>       |
| 上下文 | http, server, location              |

启用或禁用为状态大于 400 的 MSIE 客户端响应添加注释, 以增加响应大小到 512 字节。

### msie\_refresh

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>msie_refresh on   off;</code> |
| 默认值 | <code>msie_refresh off;</code>      |
| 上下文 | http, server, location              |

启用或禁用为 MSIE 客户端发出刷新而不是重定向。

### open\_file\_cache

|     |                                                                                                        |
|-----|--------------------------------------------------------------------------------------------------------|
| 语法  | <code>open_file_cache off;</code><br><code>open_file_cache max=<i>N</i> [<i>inactive=time</i>];</code> |
| 默认值 | <code>open_file_cache off;</code>                                                                      |
| 上下文 | http, server, location                                                                                 |

配置一个缓存, 可以存储:

- 打开的文件描述符, 它们的大小和修改时间;
- 目录存在的信息;
- 文件查找错误, 例如“文件未找到”, “无读取权限”等。

错误的缓存应由 `open_file_cache_errors` 指令单独启用。

|          |                                                      |
|----------|------------------------------------------------------|
| max      | 设置缓存中的最大元素数; 在缓存溢出时, 最不常用的 (LRU) 元素将被移除;             |
| inactive | 定义一个时间, 如果在此期间没有访问元素, 则将其从缓存中移除;<br>默认情况下, 设置为 60 秒。 |
| off      | 禁用缓存。                                                |

示例:

```
open_file_cache      max=1000 inactive=20s;  
open_file_cache_valid 30s;  
open_file_cache_min_uses 2;  
open_file_cache_errors on;
```

### open\_file\_cache\_errors

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>open_file_cache_errors on   off;</code> |
| 默认值 | <code>open_file_cache_errors off;</code>      |
| 上下文 | http, server, location                        |

启用或禁用由 `open_file_cache` 缓存文件查找错误。

### open\_file\_cache\_min\_uses

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>open_file_cache_min_uses number;</code> |
| 默认值 | <code>open_file_cache_min_uses 1;</code>      |
| 上下文 | http, server, location                        |

设置在 `open_file_cache` 指令的 `inactive` 参数配置的期间, 文件描述符在缓存中保持打开所需的文件访问最少 `number` 次数。

### open\_file\_cache\_valid

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>open_file_cache_valid time;</code> |
| 默认值 | <code>open_file_cache_valid 60s;</code>  |
| 上下文 | http, server, location                   |

设置 `open_file_cache` 元素应该被验证的时间。

### output\_buffers

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>output_buffers number size;</code> |
| 默认值 | <code>output_buffers 2 32k;</code>       |
| 上下文 | http, server, location                   |

设置用于从磁盘读取响应的缓冲区的 `number` 和 `size`。

### port\_in\_redirect

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>port_in_redirect on   off;</code> |
| 默认值 | <code>port_in_redirect on;</code>       |
| 上下文 | http, server, location                  |

启用或禁用在 Angie 发出的绝对重定向中指定端口。

重定向中使用主服务器名称由 `server_name_in_redirect` 指令控制。

### postpone\_output

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>postpone_output size;</code> |
| 默认值 | <code>postpone_output 1460;</code> |
| 上下文 | http, server, location             |

如果可能, 客户端数据的传输将推迟, 直到 Angie 至少有 *size* 字节的数据可发送。

|   |           |
|---|-----------|
| 0 | 禁止数据传输的推迟 |
|---|-----------|

### read\_ahead

|     |                               |
|-----|-------------------------------|
| 语法  | <code>read_ahead size;</code> |
| 默认值 | <code>read_ahead 0;</code>    |
| 上下文 | http, server, location        |

设置内核在处理文件时的预读量。

在 Linux 上, 使用 `posix_fadvise(0, 0, 0, POSIX_FADV_SEQUENTIAL)` 系统调用, 因此忽略 *size* 参数。

### recursive\_error\_pages

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>recursive_error_pages on   off;</code> |
| 默认值 | <code>recursive_error_pages off;</code>      |
| 上下文 | http, server, location                       |

启用或禁用使用 `error_page` 指令进行多次重定向。这种重定向的次数是有限的。

### request\_pool\_size

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>request_pool_size size;</code> |
| 默认值 | <code>request_pool_size 4k;</code>   |
| 上下文 | http, server                         |

允许精确调整每个请求的内存分配。该指令对性能影响最小，通常不应使用。

### reset\_timedout\_connection

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>reset_timedout_connection on   off;</code> |
| 默认值 | <code>reset_timedout_connection off;</code>      |
| 上下文 | http, server, location                           |

启用或禁用重置超时连接和用非标准代码 444 关闭的连接。重置如下进行。在关闭套接字之前，设置其 `SO_LINGER` 选项为超时值 0。当套接字关闭时，`TCP RST` 发送给客户端，并释放该套接字占用的所有内存。这有助于避免长时间保持已关闭的套接字处于 `FIN_WAIT1` 状态。

#### **i** 备注

超时的保持连接会正常关闭。

### resolver

|     |                                                                                                        |
|-----|--------------------------------------------------------------------------------------------------------|
| 语法  | <code>resolver address ... [valid=time] [ipv4=on   off] [ipv6=on   off]<br/>[status_zone=zone];</code> |
| 默认值 | —                                                                                                      |
| 上下文 | http, server, location, upstream                                                                       |

配置用于将上游服务器名称解析为地址的名称服务器，例如：

```
resolver 127.0.0.53 [::1]:5353;
```

地址可以指定为域名或 IP 地址，以及可选的端口。如果未指定端口，则使用端口 53。名称服务器以循环方式查询。

默认情况下，Angie 使用响应的 TTL 值缓存答案。

|       |                  |
|-------|------------------|
| valid | 可选参数允许重写缓存条目的有效性 |
|-------|------------------|

```
resolver 127.0.0.53 [::1]:5353 valid=30s;
```

默认情况下, Angie 在解析时将查找 IPv4 和 IPv6 地址。

|          |               |
|----------|---------------|
| ipv4=off | 禁用 IPv4 地址的查找 |
| ipv6=off | 禁用 IPv6 地址的查找 |

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| status_zone | 可选参数; 启用在指定区域中收集 DNS 服务器请求和响应指标 ( <code>/status/resolvers/&lt;zone&gt;</code> )。 |
|-------------|----------------------------------------------------------------------------------|

### 💡 小技巧

为防止 DNS 欺骗, 建议在妥善保护的可信本地网络中配置 DNS 服务器。

## resolver\_timeout

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>resolver_timeout time;</code> |
| 默认值 | <code>resolver_timeout 30s;</code>  |
| 上下文 | http, server, location, upstream    |

设置名称解析的超时时间, 例如:

```
resolver_timeout 5s;
```

## root

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>root path;</code>                |
| 默认值 | <code>root html;</code>                |
| 上下文 | http, server, location, if in location |

设置请求的根目录。例如, 使用以下配置

```
location /i/ {  
    root /data/w3;  
}
```

将会在响应 `/i/top.gif` 请求时发送 `/data/w3/i/top.gif` 文件。

路径值可以包含变量, 但不能包含 `$document_root` 和 `$realpath_root`。

文件路径是通过简单地将 URI 添加到 `root` 指令的值后构造的。如果需要修改 URI, 应使用 `alias` 指令。

### satisfy

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>satisfy all   any;</code>     |
| 默认值 | <code>satisfy all;</code>           |
| 上下文 | <code>http, server, location</code> |

允许访问如果所有 (*all*) 或至少一个 (*any*) 这些模块允许访问: 访问控制, 基础认证, 或认证请求。

```
location / {
    satisfy any;

    allow 192.168.1.0/32;
    deny all;

    auth_basic "closed site";
    auth_basic_user_file conf/htpasswd;
}
```

### send\_lowat

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>send_lowat size;</code>       |
| 默认值 | <code>send_lowat 0;</code>          |
| 上下文 | <code>http, server, location</code> |

如果指令设置为非零值, Angie 将尝试通过使用 `NOTE_LOWAT` 标志的 `kqueue` 方法或 `SO_SNDLOWAT` 套接字选项来最小化客户端套接字上的发送操作次数。在这两种情况下, 使用指定的大小。

### send\_timeout

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>send_timeout time;</code>     |
| 默认值 | <code>send_timeout 60s;</code>      |
| 上下文 | <code>http, server, location</code> |

设置向客户端发送响应的超时时间。超时时间仅设置在两个连续的写操作之间, 而不是整个响应的发送时间。如果客户端在此时间内没有收到任何信息, 连接将被关闭。



## sendfile

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>sendfile on   off;</code>        |
| 默认值 | <code>sendfile off;</code>             |
| 上下文 | http, server, location, if in location |

启用或禁用 `sendfile()` 的使用。

`aio` 可以用于预加载 `sendfile()` 的数据:

```
location /video/ {
    sendfile      on;
    tcp_nopush   on;
    aio           on;
}
```

在此配置中, `sendfile()` 以 `SF_NODISKIO` 标志调用, 这会导致它不会阻塞在磁盘 I/O 上, 而是报告数据不在内存中。然后, Angie 通过读取一个字节来启动异步数据加载。在第一次读取时, FreeBSD 内核将文件的前 128K 字节加载到内存中, 尽管接下来的读取将仅以 16K 块加载数据。这可以通过 `read_ahead` 指令进行更改。

## sendfile\_max\_chunk

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>sendfile_max_chunk size;</code> |
| 默认值 | <code>sendfile_max_chunk 2m;</code>   |
| 上下文 | http, server, location                |

限制单个 `sendfile()` 调用中可以传输的数据量。没有限制的情况下, 一个快速连接可能会完全占用工作进程。

## server

|     |                             |
|-----|-----------------------------|
| 语法  | <code>server { ... }</code> |
| 默认值 | —                           |
| 上下文 | http                        |

设置虚拟服务器的配置。IP 地址 (基于 IP 地址) 和名称 (基于 “Host” 请求头字段) 之间没有明确的分隔。相反, `listen` 指令描述了应该接受连接的所有地址和端口, 而 `server_name` 指令列出了所有服务器名称。

示例配置在 *How Angie processes a request* 文档中提供。

## server\_name

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>server_name name ...;</code> |
| 默认值 | <code>server_name "";</code>       |
| 上下文 | server                             |

设置虚拟服务器的名称, 例如:

```
server {
    server_name example.com www.example.com;
}
```

第一个名称成为主要服务器名称。

服务器名称可以包含用星号 (“\*”) 替换名称的第一个或最后一个部分:

```
server {
    server_name example.com *.example.com www.example.*;
}
```

这些名称被称为通配符名称。

上述名称中的前两个可以合并为一个:

```
server {
    server_name .example.com;
}
```

还可以在服务器名称中使用正则表达式, 在名称前加上波浪号 (“~”):

```
server {
    server_name ~^www\d+\.example\.com$ www.example.com;
}
```

正则表达式可以包含捕获, 稍后可以在其他指令中使用:

```
server {
    server_name ~^(www\.)?(.+)$;

    location / {
        root /sites/$2;
    }
}

server {
    server_name _;

    location / {
```

```
    root /sites/default;
}
}
```

在正则表达式中命名捕获会创建变量稍后可以在其他指令中使用:

```
server {
    server_name ~^(www\.)?(?<domain>.+)$;

    location / {
        root /sites/$domain;
    }
}

server {
    server_name _;

    location / {
        root /sites/default;
    }
}
```

#### 备注

如果指令设置为 `$hostname`, 则使用 Web 服务器的主机名。

您还可以指定一个空服务器名称 (""):

```
server {
    server_name www.example.com "";
}
```

在按名称搜索虚拟服务器时, 如果匹配多个选项 (例如, 通配符和正则表达式), 将按以下优先顺序选择第一个匹配项:

- 精确名称;
- 以通配符开头的最长名称, 例如 `*.example.com`;
- 以通配符结尾的最长名称, 例如 `mail.*`;
- 第一个匹配的正则表达式 (按出现顺序), 包括一个空名称。

#### 注意

要使 `server_name` 在 TLS 中工作, 您需要终止 TLS 连接。该指令匹配 HTTP 请求中的 `Host`, 因此必须完成握手并解密连接。

### server\_name\_in\_redirect

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>server_name_in_redirect on   off;</code> |
| 默认值 | <code>server_name_in_redirect off;</code>      |
| 上下文 | http, server, location                         |

启用或禁用用在 `Angie` 发出的绝对重定向中使用由 `server_name` 指令指定的主要服务器名称。

|     |                                            |
|-----|--------------------------------------------|
| on  | 由 <code>server_name</code> 指令指定的主要服务器名称    |
| off | 使用“Host”请求头字段中的名称。如果该字段不存在, 则使用服务器的 IP 地址。 |

重定向中端口的使用由 `port_in_redirect` 指令控制。

### server\_names\_hash\_bucket\_size

|     |                                                           |
|-----|-----------------------------------------------------------|
| 语法  | <code>server_names_hash_bucket_size size;</code>          |
| 默认值 | <code>server_names_hash_bucket_size 32   64   128;</code> |
| 上下文 | http                                                      |

设置服务器名称哈希表的桶大小。默认值取决于处理器缓存行的大小。有关设置哈希表的详细信息, 请参阅单独的 文档。

### server\_names\_hash\_max\_size

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>server_names_hash_max_size size;</code> |
| 默认值 | <code>server_names_hash_max_size 512;</code>  |
| 上下文 | http                                          |

设置服务器名称哈希表的最大大小。有关设置哈希表的详细信息, 请参阅单独的 文档。

### server\_tokens

|     |                                                       |
|-----|-------------------------------------------------------|
| 语法  | <code>server_tokens on   off   build   string;</code> |
| 默认值 | <code>server_tokens on;</code>                        |
| 上下文 | http, server, location                                |

启用或禁用 在错误页面和 Server 响应头字段中显示 Angie 版本。 *build* 参数启用显示构建名称, 由相应的 *configure* 参数设置, 以及版本。

Added in version 1.1.0: PRO

在 Angie PRO 中, 如果指令设置了一个 *string*, 其中可能还包含变量, 错误页面和 Server 响应头字段将使用字符串的变量插值值而不是服务器名称、版本和构建名称。空的 *string* 禁用显示 Server 字段。

### status\_zone

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>status_zone zone;</code>   |
| 默认值 | —                                |
| 上下文 | server, location, if in location |

分配一个共享内存区域以收集指标用于 `/status/http/location_zones/<zone>` 和 `/status/http/server_zones/<zone>`。

多个 server 上下文可以共享相同的区域进行数据收集; 特殊值 `off` 在嵌套的 location 块中禁用数据收集。

### subrequest\_output\_buffer\_size

|     |                                                     |
|-----|-----------------------------------------------------|
| 语法  | <code>subrequest_output_buffer_size size;</code>    |
| 默认值 | <code>subrequest_output_buffer_size 4k   8k;</code> |
| 上下文 | http, server, location                              |

设置用于存储子请求响应体的缓冲区大小。默认情况下, 缓冲区大小等于一个内存页。这是 4K 或 8K, 具体取决于平台。可以将其设置得更小, 然而。

#### **i** 备注

该指令仅适用于将响应体保存到内存中的子请求。例如, *SSI* 创建的子请求。

### tcp\_nodelay

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>tcp_nodelay on   off;</code> |
| 默认值 | <code>tcp_nodelay on;</code>       |
| 上下文 | http, server, location             |

启用或禁用使用 `TCP_NODELAY` 选项。当连接进入保持活动状态时, 该选项被启用。此外, 它在 SSL 连接、无缓冲代理以及 `WebSocket` 代理 中被启用。

### tcp\_nopush

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>tcp_nopush on   off;</code>   |
| 默认值 | <code>tcp_nopush off;</code>        |
| 上下文 | <code>http, server, location</code> |

启用或禁用在 FreeBSD 上的 `TCP_NOPUSH` 套接字选项或在 Linux 上的 `TCP_CORK` 套接字选项。仅在使用 `sendfile` 时启用这些选项。启用该选项可以:

- 在 Linux 和 FreeBSD 4.\* 上, 在一个数据包中发送响应头和文件的开头;
- 以完整数据包发送文件。

### try\_files

|     |                                                                                |
|-----|--------------------------------------------------------------------------------|
| 语法  | <code>try_files file ... uri;</code><br><code>try_files file ... =code;</code> |
| 默认值 | —                                                                              |
| 上下文 | <code>server, location</code>                                                  |

检查指定顺序的文件是否存在, 并使用第一个找到的文件进行请求处理; 处理在当前上下文中执行。文件路径根据 `root` 和 `alias` 指令从文件参数构建。可以通过在名称末尾指定斜杠来检查目录的存在, 例如 `$uri/`。如果未找到任何文件, 则会内部重定向到最后一个参数中指定的 `uri`。例如:

```
location /images/ {
    try_files $uri /images/default.gif;
}

location = /images/default.gif {
    expires 30s;
}
```

最后一个参数也可以指向一个命名的 `location`, 如下例所示。最后一个参数也可以是一个代码:

```
location / {
    try_files $uri $uri/index.html $uri.html =404;
}
```

在以下示例中,

```
location / {
    try_files $uri $uri/ @drupal;
}
```

*try\_files* 指令等同于

```
location / {
    error_page 404 = @drupal;
    log_not_found off;
}
```

如这里,

```
location ~ /\.php$ {
    try_files $uri @drupal;

    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;

# ...
}
```

*try\_files* 在将请求传递给 FastCGI 服务器之前检查 PHP 文件的存在。

```
location / {
    try_files /system/maintenance.html
             $uri $uri/index.html $uri.html
             @mongrel;
}

location @mongrel {
    proxy_pass http://mongrel;
}
```

```
location / {
    try_files $uri $uri/ @drupal;
}

location ~ /\.php$ {
    try_files $uri @drupal;

    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
    fastcgi_param SCRIPT_NAME     $fastcgi_script_name;
    fastcgi_param QUERY_STRING    $args;
}
```

```
# ... other fastcgi_param
}

location @drupal {
    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to/index.php;
    fastcgi_param SCRIPT_NAME     /index.php;
    fastcgi_param QUERY_STRING    q=$uri&$args;

# ... other fastcgi_param
}
```

```
location / {
    try_files $uri $uri/ @wordpress;
}

location ~ /\.php$ {
    try_files $uri @wordpress;

    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to$fastcgi_script_name;
# ... other fastcgi_param
}

location @wordpress {
    fastcgi_pass ...;

    fastcgi_param SCRIPT_FILENAME /path/to/index.php;
# ... other fastcgi_param
}
```

## types

|     |                                                      |
|-----|------------------------------------------------------|
| 语法  | types { ... }                                        |
| 默认值 | types text/html html; image/gif gif; image/jpeg jpg; |
| 上下文 | http, server, location                               |

将文件扩展名映射到响应的 MIME 类型。扩展名不区分大小写。多个扩展名可以映射到一个类型，例如：

```
types {
    application/octet-stream bin exe dll;
    application/octet-stream deb;
    application/octet-stream dmg;
```



```
}
```

一份足够完整的映射表随 Angie 一同发布在 `conf/mime.types` 文件中。

为了使特定位置对所有请求发出“`application/octet-stream`” MIME 类型, 可以使用以下配置:

```
location /download/ {  
    types          { }  
    default_type application/octet-stream;  
}
```

### types\_hash\_bucket\_size

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>types_hash_bucket_size size;</code> |
| 默认值 | <code>types_hash_bucket_size 64;</code>   |
| 上下文 | http, server, location                    |

设置类型哈希表的桶大小。有关设置哈希表的详细信息, 请参见单独的 文档。

### types\_hash\_max\_size

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>types_hash_max_size size;</code> |
| 默认值 | <code>types_hash_max_size 1024;</code> |
| 上下文 | http, server, location                 |

设置类型哈希表的最大大小。有关设置哈希表的详细信息, 请参见单独的 文档。

### underscores\_in\_headers

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>underscores_in_headers on   off;</code> |
| 默认值 | <code>underscores_in_headers off;</code>      |
| 上下文 | http, server                                  |

启用或禁用在客户端请求头字段中使用下划线。当禁用下划线使用时, 名称包含下划线的请求头字段将被标记为无效, 并受制于 `ignore_invalid_headers` 指令。

如果在 `server` 级别指定了该指令, 则可以使用默认服务器的值。

### variables\_hash\_bucket\_size

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>variables_hash_bucket_size size;</code> |
| 默认值 | <code>variables_hash_bucket_size 64;</code>   |
| 上下文 | http                                          |

设置变量哈希表的桶大小。有关设置哈希表的详细信息，请参见单独的 文档。

### variables\_hash\_max\_size

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>variables_hash_max_size size;</code> |
| 默认值 | <code>variables_hash_max_size 1024;</code> |
| 上下文 | http                                       |

设置变量哈希表的最大大小。有关设置哈希表的详细信息，请参见单独的 文档。

## 内置变量

`http_core` 模块支持名称与 Apache 服务器变量匹配的内置变量。首先，这些是表示客户端请求头字段的变量，例如 `$http_user_agent`、`$http_cookie` 等。此外，还有其他变量：

### `$angie_version`

Angie 版本

### `$arg_<name>`

请求行中具有指定 *name* 的参数

### `$args`

请求行中的参数

`$binary_remote_addr`

以二进制形式表示的客户端地址, 值的长度对于 IPv4 地址始终为 4 字节, IPv6 地址为 16 字节

`$body_bytes_sent`

发送给客户端的字节数, 不包括响应头; 此变量与 Apache 模块 `mod_log_config` 的”%B” 参数兼容

`$bytes_sent`

发送给客户端的字节数

`$connection`

连接序列号

`$connection_requests`

通过连接发起的当前请求数

`$connection_time`

连接时间 (以秒为单位, 精确到毫秒)

`$content_length`

”Content-Length” 请求头字段

`$content_type`

”Content-Type” 请求头字段

`$cookie_<name>`

具有指定 *name* 的 cookie

`$document_root`

当前请求的 *root* 或 *alias* 指令的值

`$document_uri`

与 *\$uri* 相同

`$host`

按此优先顺序: 请求行中的主机名, 或 "Host" 请求头字段中的主机名, 或匹配请求的服务器名

`$hostname`

主机名

`$http_<name>`

任意请求头字段; *name* 是字段名称转为小写并将破折号替换为下划线

`$https`

如果连接在 SSL 模式下操作, 则为 *on*, 否则为空字符串

`$is_args`

如果请求行有参数则为 *?*, 否则为空字符串

`$limit_rate`

设置此变量以启用响应速率限制; 参见 *limit\_rate*

`$msec`

当前时间 (以秒为单位, 精确到毫秒)

`$pid`

工作进程的 PID

`$pipe`

如果请求是流水线的, 则为 `p`, 否则为 `.`

`$proxy_protocol_addr`

来自 PROXY 协议头的客户端地址。

必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

`$proxy_protocol_port`

来自 PROXY 协议头的客户端端口。

必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

`$proxy_protocol_server_addr`

来自 PROXY 协议头的服务器地址。

必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

`$proxy_protocol_server_port`

来自 PROXY 协议头的服务器端口。

必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

`$proxy_protocol_tlv_<name>`

来自 PROXY 协议头的 TLV。 `name` 可以是 TLV 类型或其数值。在后一种情况下, 值是十六进制的, 并应以 `0x` 为前缀:

```
$proxy_protocol_tlv_alpn
$proxy_protocol_tlv_0x01
```

SSL TLV 也可以通过 TLV 类型名称或其数值来访问, 这两者都以 `ssl_` 为前缀:

```
$proxy_protocol_tlv_ssl_version
$proxy_protocol_tlv_ssl_0x21
```

支持以下 TLV 类型名称:

- *alpn (0x01)* - 连接上使用的上层协议
- *authority (0x02)* - 客户端传递的主机名值
- *unique\_id (0x05)* - 唯一连接 ID
- *netns (0x30)* - 命名空间名称
- *ssl (0x20)* - 二进制 SSL TLV 结构

支持以下 SSL TLV 类型名称:

- *ssl\_version (0x21)* - 客户端连接中使用的 SSL 版本
- *ssl\_cn (0x22)* - SSL 证书的通用名称
- *ssl\_cipher (0x23)* - 使用的密码名称
- *ssl\_sig\_alg (0x24)* - 用于签署证书的算法
- *ssl\_key\_alg (0x25)* - 公钥算法

还支持以下特殊的 SSL TLV 类型名称:

- *ssl\_verify* - 客户端 SSL 证书验证结果, 如果客户端提供了证书并成功验证则为 '0', 否则为非零。

PROXY 协议必须在 *listen* 指令中设置 *proxy\_protocol* 参数以预先启用。

`$query_string`

同 `$args`

`$realpath_root`

对应于当前请求的 *root* 或 *alias* 指令值的绝对路径名, 所有符号链接都解析为真实路径

`$remote_addr`

客户端地址

`$remote_port`

客户端端口

`$remote_user`

通过基本认证提供的用户名

`$request`

完整的原始请求行

`$request_body`

请求体。

当请求体被读入 *memory buffer* 时, 该变量的值在由 *proxy\_pass*、*fastcgi\_pass*、*uwsgi\_pass* 和 *scgi\_pass* 指令处理的位置中 \* 可用 \*。

`$request_body_file`

请求体的临时文件名。

处理结束时, 需要删除该文件。若要始终将请求体写入文件, 需要启用 *client\_body\_in\_file\_only*。当在代理请求或 FastCGI/uwsgi/SCGI 服务器请求中传递临时文件名时, 应通过 *proxy\_pass\_request\_body off*、*fastcgi\_pass\_request\_body off*、*uwsgi\_pass\_request\_body off* 或 *scgi\_pass\_request\_body off* 指令禁用请求体的传递。

`$request_completion`

若请求已完成则为“OK”, 否则为空字符串

`$request_filename`

基于 *root* 或 *alias* 指令和请求 URI 的当前请求的文件路径

`$request_id`

由 16 个随机字节生成的唯一请求标识符, 以十六进制表示

`$request_length`

请求长度 (包括请求行、头部和请求体)

`$request_method`

请求方法, 通常为 GET 或 POST

`$request_time`

请求处理时间, 以秒为单位, 精确到毫秒; 从客户端读取第一个字节起的时间

`$request_uri`

完整的原始请求 URI (带参数)

`$scheme`

请求方案, "http" 或 "https"

`$sent_http_<name>`

任意响应头字段; *name* 是字段名称, 转换为小写并用下划线替换破折号

`$sent_trailer_<name>`

响应结尾发送的任意字段; *name* 是字段名称, 转换为小写并用下划线替换破折号

`$server_addr`

接受请求的服务器地址。计算该变量的值通常需要一个系统调用。为了避免它, *listen* 指令必须指定地址并使用 *bind* 参数。



`$server_name`

接受请求的服务器名称

`$server_port`

接受请求的服务器端口

`$server_protocol`

请求协议, 通常为"HTTP/1.0"、"HTTP/1.1" 或"HTTP/2.0"

`$status`

响应状态

`$time_iso8601`

ISO 8601 标准格式的本地时间

`$time_local`

常用日志格式的本地时间

`$tcpinfo_rtt`, `$tcpinfo_rttvar`, `$tcpinfo_snd_cwnd`, `$tcpinfo_rcv_space`

有关客户端 TCP 连接的信息; 在支持 TCP\_INFO 套接字选项的系统上可用

`$uri`

请求中的当前 URI, *normalized*。在请求处理过程中, `$uri` 的值可能会更改, 例如在执行内部重定向或使用索引文件时。

### 3.3.3 流模块

#### 访问

该模块允许限制某些客户端地址的访问。

#### 配置示例

```
server {  
    ...  
    deny 192.168.1.1;  
    allow 192.168.1.0/24;  
    allow 10.1.1.0/16;  
    allow 2001:0db8::/32;  
    deny all;  
}
```

规则按顺序检查，直到找到第一个匹配项。在此示例中，只有 IPv4 网络 *10.1.1.0/16* 和 *192.168.1.0/24* (不包括地址 *192.168.1.1*) 以及 IPv6 网络 *2001:0db8::/32* 允许访问。

#### 指令

##### allow

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>allow address   CIDR   unix:   all;</code> |
| 默认值 | —                                                |
| 上下文 | http, server, location, limit_except             |

允许指定网络或地址的访问。如果指定了特殊值 `unix:` (1.5.1)，则允许所有 UNIX 域套接字的访问。

##### deny

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>deny address   CIDR   unix:   all;</code> |
| 默认值 | —                                               |
| 上下文 | http, server, location, limit_except            |

拒绝指定网络或地址的访问。如果指定了特殊值 `unix:` (1.5.1)，则拒绝所有 UNIX 域套接字的访问。

## Geo

该模块根据客户端 IP 地址创建具有不同值的变量。

### 配置示例

```
geo $geo {  
    default      0;  
  
    127.0.0.1    2;  
    192.168.1.0/24 1;  
    10.1.0.0/16  1;  
  
    ::1         2;  
    2001:0db8::/32 1;  
}
```

## 指令

### geo

|     |                                                               |
|-----|---------------------------------------------------------------|
| 语法  | <code>geo [<i>\$address</i>] <i>\$variable</i> { ... }</code> |
| 默认  | —                                                             |
| 上下文 | stream                                                        |

描述指定变量的值如何依赖于客户端 IP 地址。默认情况下, 地址取自 `$remote_addr` 变量, 但也可以取自其他变量, 例如:

```
geo $arg_remote_addr $geo {  
    ...;  
}
```

#### 备注

因为变量仅在使用时才被评估, 所以即使存在大量声明的 `geo` 变量, 也不会对连接处理造成额外开销。

如果变量的值不是一个有效的 IP 地址, 则使用“255.255.255.255”地址。

地址可以指定为 CIDR 表示法中的前缀 (包括单个地址) 或作为范围。

还支持以下特殊参数:

|                      |                                                                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>delete</code>  | 删除指定的网络                                                                                                                                                       |
| <code>default</code> | 当客户端地址不匹配任何指定地址时, 赋给变量的值。当地址以 CIDR 表示法指定时, 可以用“ <code>0.0.0.0/0</code> ”和“ <code>::/0</code> ”代替 <code>default</code> 。当未指定 <code>default</code> 时, 默认值为空字符串。 |
| <code>include</code> | 包含一个包含地址和值的文件。可以有多个包含。                                                                                                                                        |
| <code>ranges</code>  | 指示地址以范围形式指定。此参数应放在首位。为了加快 geo 基础的加载, 地址应按升序排列。                                                                                                                |

示例:

```
geo $country {
    default      ZZ;
    include      conf/geo.conf;
    delete      127.0.0.0/16;
    proxy        192.168.100.0/24;
    proxy        2001:0db8::/32;

    127.0.0.0/24 US;
    127.0.0.1/32 RU;
    10.1.0.0/16  RU;
    192.168.1.0/24 UK;
}
```

`conf/geo.conf` 文件可能包含以下行:

```
10.2.0.0/16  RU;
192.168.2.0/24 RU;
```

使用最具体匹配的值。例如, 对于 `127.0.0.1` 地址, 将选择值 `RU`, 而不是 `US`。

使用范围的示例:

```
geo $country {
    ranges;
    default      ZZ;
    127.0.0.0-127.0.0.0  US;
    127.0.0.1-127.0.0.1  RU;
    127.0.0.2-127.0.0.255 US;
    10.1.0.0-10.1.255.255  RU;
    192.168.1.0-192.168.1.255 UK;
}
```

## GeoIP

使用预编译的 MaxMind 数据库, 根据客户端 IP 地址创建变量。

当使用支持 IPv6 的数据库时, IPv4 地址会被查找为 IPv4 映射的 IPv6 地址。

在从源代码构建时, 此模块默认不构建; 需要使用 `--with-stream_geoip_module` 构建选项启用。

### 重要

此模块需要 MaxMind GeoIP 库。

## 配置示例

```
stream {
    geoip_country      GeoIP.dat;
    geoip_city         GeoLiteCity.dat;

    map $geoip_city_continent_code $nearest_server {
        default        example.com;
        EU              eu.example.com;
        NA              na.example.com;
        AS              as.example.com;
    }
# ...
}
```

## 指令

### geoip\_country

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>geoip_country file;</code> |
| 默认值 | —                                |
| 上下文 | stream                           |

指定用于根据客户端 IP 地址确定国家的数据库。使用此数据库时可以使用以下变量:

`$geoip_country_c` 两字母国家代码, 例如, “RU”, “US”。

`$geoip_country_c` 三字母国家代码, 例如, “RUS”, “USA”。

`$geoip_country_n` 国家名称, 例如, “Russian Federation”, “United States”。

## geoip\_city

|     |                               |
|-----|-------------------------------|
| 语法  | <code>geoip_city file;</code> |
| 默认值 | —                             |
| 上下文 | stream                        |

指定用于根据客户端 IP 地址确定国家、地区和城市的数据库。使用此数据库时可以使用以下变量：

|                                  |                                                                  |
|----------------------------------|------------------------------------------------------------------|
| <code>\$geoip_city_cont</code>   | 两字母洲代码，例如，“EU”，“NA”。                                             |
| <code>\$geoip_city_coun</code>   | 两字母国家代码，例如，“RU”，“US”。                                            |
| <code>\$geoip_city_coun</code>   | 三字母国家代码，例如，“RUS”，“USA”。                                          |
| <code>\$geoip_city_coun</code>   | 国家名称，例如，“Russian Federation”，“United States”。                    |
| <code>\$geoip_dma_code</code>    | 美国的 DMA 区域代码（也称为“地铁代码”），根据 Google AdWords API 中的地理定位。            |
| <code>\$geoip_latitude</code>    | 纬度。                                                              |
| <code>\$geoip_longitude</code>   | 经度。                                                              |
| <code>\$geoip_region</code>      | 两个字符的国家地区代码（地区、领土、州、省、联邦土地等），例如，“48”，“DC”。                       |
| <code>\$geoip_region_name</code> | 国家地区名称（地区、领土、州、省、联邦土地等），例如，“Moscow City”，“District of Columbia”。 |
| <code>\$geoip_city</code>        | 城市名称，例如，“Moscow”，“Washington”。                                   |
| <code>\$geoip_postal_code</code> | 邮政编码。                                                            |

## geoip\_org

|     |                              |
|-----|------------------------------|
| 语法  | <code>geoip_org file;</code> |
| 默认值 | —                            |
| 上下文 | stream                       |

指定用于根据客户端 IP 地址确定组织的数据库。使用此数据库时可以使用以下变量：

|                          |                                        |
|--------------------------|----------------------------------------|
| <code>\$geoip_org</code> | 组织名称，例如，“The University of Melbourne”。 |
|--------------------------|----------------------------------------|

## JS

该模块用于在 njs 中实现处理程序——njs 是 JavaScript 语言的一个子集。

在我们的代码库中, 该模块是动态构建的并作为一个名为 `angie-module-njs` 或 `angie-pro-module-njs` 的独立包提供。

### 配置示例

```
stream {
    js_import stream.js;

    js_set $bar stream.bar;
    js_set $req_line stream.req_line;

    server {
        listen 12345;

        js_preread stream.preread;
        return $req_line;
    }

    server {
        listen 12346;

        js_access stream.access;
        proxy_pass 127.0.0.1:8000;
        js_filter stream.header_inject;
    }
}

http {
    server {
        listen 8000;
        location / {
            return 200 $http_foo\n;
        }
    }
}
```

stream.js 文件内容:

```
var line = '';

function bar(s) {
    var v = s.variables;
    s.log("hello from bar() handler!");
    return "bar-var" + v.remote_port + "; pid=" + v.pid;
}
```

```
}

function preread(s) {
  s.on('upload', function (data, flags) {
    var n = data.indexOf('\n');
    if (n !== -1) {
      line = data.substr(0, n);
      s.done();
    }
  });
}

function req_line(s) {
  return line;
}

// 读取 HTTP 请求行。
// 收集 'req' 中的字节, 直到
// 请求行被读取。
// 将 HTTP 头注入客户端请求

var my_header = 'Foo: foo';
function header_inject(s) {
  var req = '';
  s.on('upload', function(data, flags) {
    req += data;
    var n = req.search('\n');
    if (n !== -1) {
      var rest = req.substr(n + 1);
      req = req.substr(0, n + 1);
      s.send(req + my_header + '\r\n' + rest, flags);
      s.off('upload');
    }
  });
}

function access(s) {
  if (s.remoteAddress.match('^192.*')) {
    s.deny();
    return;
  }

  s.allow();
}

export default {bar, preread, req_line, header_inject, access};
```



## 指令

### js\_access

|     |                                                    |
|-----|----------------------------------------------------|
| 语法  | <code>js_access function   module.function;</code> |
| 默认值 | —                                                  |
| 上下文 | stream, server                                     |

设置将在访问阶段调用的 `njs` 函数。可以引用模块函数。

当流会话首次达到访问阶段时, 该函数会被调用一次。该函数使用以下参数调用:

|                |                   |
|----------------|-------------------|
| <code>s</code> | stream session 对象 |
|----------------|-------------------|

在此阶段, 可以使用 `s.on()` 方法为每个传入的数据块执行初始化或注册回调, 直到调用以下方法之一: `s.done()`, `s.decline()`, `s.allow()`。一旦调用了这些方法之一, 流会话处理将切换到下一个阶段, 并且所有当前的 `s.on()` 回调将被丢弃。

### js\_fetch\_buffer\_size

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>js_fetch_buffer_size size;</code> |
| 默认值 | <code>js_fetch_buffer_size 16k;</code>  |
| 上下文 | stream, server                          |

设置用于读取和写入 Fetch API 的缓冲区大小。

### js\_fetch\_ciphers

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>js_fetch_ciphers ciphers;</code>          |
| 默认值 | <code>js_fetch_ciphers HIGH:!aNULL:!MD5;</code> |
| 上下文 | stream, server                                  |

指定 Fetch API 的 HTTPS 连接中启用的加密套件。这些加密套件以 OpenSSL 库理解的格式指定。

可使用“`openssl ciphers`”命令查看完整列表。

### js\_fetch\_max\_response\_buffer\_size

|     |                                                      |
|-----|------------------------------------------------------|
| 语法  | <code>js_fetch_max_response_buffer_size size;</code> |
| 默认值 | <code>js_fetch_max_response_buffer_size 1m;</code>   |
| 上下文 | stream, server                                       |

设置使用 Fetch API 接收的响应的最大大小。

### js\_fetch\_protocols

|     |                                                                        |
|-----|------------------------------------------------------------------------|
| 语法  | <code>js_fetch_protocols [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];</code> |
| 默认值 | <code>js_fetch_protocols TLSv1 TLSv1.1 TLSv1.2;</code>                 |
| 上下文 | stream, server                                                         |

启用 Fetch API 的 HTTPS 连接的指定协议。

### js\_fetch\_timeout

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>js_fetch_timeout time;</code> |
| 默认值 | <code>js_fetch_timeout 60s;</code>  |
| 上下文 | stream, server                      |

定义 Fetch API 的读取和写入超时时间。该超时时间仅在两次连续的读/写操作之间设置，而不是针对整个响应。如果在此时间内没有传输数据，连接将被关闭。

### js\_fetch\_trusted\_certificate

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>js_fetch_trusted_certificate file;</code> |
| 默认值 | —                                               |
| 上下文 | stream, server                                  |

指定一个包含 PEM 格式的受信 CA 证书的文件，用于使用 Fetch API 验证 HTTPS 证书。

### js\_fetch\_verify

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>js_fetch_verify on   off;</code> |
| 默认值 | <code>js_fetch_verify on;</code>       |
| 上下文 | stream, server                         |

启用或禁用使用 Fetch API 验证 HTTPS 服务器证书。

### js\_fetch\_verify\_depth

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>js_fetch_verify_depth number;</code> |
| 默认值 | <code>js_fetch_verify_depth 100;</code>    |
| 上下文 | stream, server                             |

设置 Fetch API 的 HTTPS 服务器证书链的验证深度。

### js\_filter

|     |                                                    |
|-----|----------------------------------------------------|
| 语法  | <code>js_filter function   module.function;</code> |
| 默认值 | —                                                  |
| 上下文 | stream, server                                     |

设置数据过滤器。可以引用模块函数。

当流会话达到内容阶段时, 过滤器函数会被调用一次。过滤器函数使用以下参数调用:

|   |                   |
|---|-------------------|
| s | stream session 对象 |
|---|-------------------|

在此阶段, 可以使用 `s.on()` 方法为每个传入的数据块执行初始化或注册回调。可以使用 `s.off()` 方法注销回调并停止过滤。

#### **i** 备注

由于 `js_filter` 处理程序立即返回其结果, 因此仅支持同步操作。因此, 不支持异步操作, 如 `ngx.fetch()` 或 `setTimeout()`。

### js\_import

|     |                                                                |
|-----|----------------------------------------------------------------|
| 语法  | <code>js_import module.js   export_name from module.js;</code> |
| 默认值 | —                                                              |
| 上下文 | stream, server                                                 |

导入在 `njs` 中实现位置和变量处理程序的模块。`export_name` 用作访问模块函数的命名空间。如果未指定 `export_name`, 则使用模块名称作为命名空间。

```
js_import stream.js;
```

在此, 模块名称 `stream` 被用作

可以指定多个 `js_import` 指令。

### js\_path

|     |                            |
|-----|----------------------------|
| 语法  | <code>js_path path;</code> |
| 默认值 | —                          |
| 上下文 | stream, server             |

设置 `njs` 模块的附加路径。

### js\_preload\_object

|     |                                                                 |
|-----|-----------------------------------------------------------------|
| 语法  | <code>js_preload_object name.json   name from file.json;</code> |
| 默认值 | —                                                               |
| 上下文 | stream, server                                                  |

在配置时预加载一个不可变对象。`name` 用作在 `njs` 代码中通过该名称访问对象的全局变量名。如果未指定 `name`, 则使用文件名。

```
js_preload_object map.json;
```

在此, `map` 作为访问预加载对象时使用的名称。

可以指定多个 `js_preload_object` 指令。

## js\_preread

|     |                                                     |
|-----|-----------------------------------------------------|
| 语法  | <code>js_preread function   module.function;</code> |
| 默认值 | —                                                   |
| 上下文 | stream, server                                      |

设置将在预读取阶段调用的 `njs` 函数。可以引用模块函数。

当流会话首次达到预读取阶段时, 该函数会被调用一次。该函数使用以下参数调用:

|                |                   |
|----------------|-------------------|
| <code>s</code> | stream session 对象 |
|----------------|-------------------|

在这个阶段, 可以为每个接收到的数据块执行初始化或使用 `s.on()` 方法注册回调, 直到调用以下方法之一: `s.done()`、`s.decline()`、`s.allow()`。当调用这些方法之一时, 流会话将切换到下一个阶段, 并且当前所有的 `s.on()` 回调将被丢弃。

### **i** 备注

因为 `js_preread` 处理器会立即返回其结果, 所以它仅支持同步回调。因此, 不支持像 `ngx.fetch()` 或 `setTimeout()` 这样的异步回调。不过, 在预读阶段的 `s.on()` 回调中支持异步操作。

## js\_set

|     |                                                            |
|-----|------------------------------------------------------------|
| 语法  | <code>js_set \$variable function   module.function;</code> |
| 默认值 | —                                                          |
| 上下文 | stream, server                                             |

为指定变量设置 `njs` 函数。可以引用模块函数。

当第一次引用给定请求的变量时调用该函数。具体时机取决于引用变量的阶段。这可以用于执行一些与变量评估无关的逻辑。例如, 如果变量仅在 `log_format` 指令中引用, 其处理器将在日志阶段之前不会被执行。这个处理器可以用于在请求释放之前进行一些清理。

### **i** 备注

因为 `js_set` 处理器会立即返回其结果, 所以它仅支持同步回调。因此, 不支持像 `ngx.fetch()` 或 `setTimeout()` 这样的异步回调。

## js\_shared\_dict\_zone

|     |                                                                                                |
|-----|------------------------------------------------------------------------------------------------|
| 语法  | <code>js_shared_dict_zone zone=name:size [timeout=time] [type=string   number] [evict];</code> |
| 默认值 | —                                                                                              |
| 上下文 | stream                                                                                         |

设置共享内存区域的名称和大小, 该区域在工作进程之间共享键值字典。

|         |                                                                                 |
|---------|---------------------------------------------------------------------------------|
| type    | 可选参数, 允许将值类型重新定义为 <code>number</code> , 默认情况下, 共享字典使用 <code>string</code> 作为键和值 |
| timeout | 可选参数, 设置从区域中移除所有共享字典条目的时间                                                       |
| evict   | 可选参数, 当区域存储耗尽时, 移除最旧的键值对                                                        |

示例:

example.conf:

```
# 创建一个1Mb的字典, 值为字符串,  
# 在60秒不活动后移除键值对:  
js_shared_dict_zone zone=foo:1M timeout=60s;  
  
# 创建一个512Kb的字典, 值为字符串,  
# 当区域耗尽时强制移除最旧的键值对:  
js_shared_dict_zone zone=bar:512K timeout=30s evict;  
  
# 创建一个32Kb的永久字典, 值为数字:  
js_shared_dict_zone zone=num:32k type=number;
```

example.js:

```
function get(r) {  
    r.return(200, ngx.shared.foo.get(r.args.key));  
}  
  
function set(r) {  
    r.return(200, ngx.shared.foo.set(r.args.key, r.args.value));  
}  
  
function delete(r) {  
    r.return(200, ngx.shared.bar.delete(r.args.key));  
}  
  
function increment(r) {  
    r.return(200, ngx.shared.num.incr(r.args.key, 2));  
}
```

## js\_var

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>js_var \$variable [value];</code> |
| 默认值 | —                                       |
| 上下文 | stream, server                          |

声明一个 可写 变量。值可以包含文本、变量及其组合。

## 会话对象属性

每个流 `njs` 处理器接收一个参数, 即 流会话 对象。

## 连接限制

该模块用于按照定义的键限制每个键的连接数, 特别是限制来自单个 IP 地址的连接数。

## 配置示例

```
stream {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        limit_conn          addr 1;
        limit_conn_log_level error;
    }
}
```

## 指令

### limit\_conn

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>limit_conn zone number;</code> |
| 默认值 | —                                    |
| 上下文 | stream, server                       |

设置共享内存区域以及给定键值的最大允许连接数。当超过此限制时, 服务器将关闭连接。例如, 指令

```
limit_conn_zone $binary_remote_addr zone=addr:10m;

server {
    ...
    limit_conn addr 1;
}
```

允许每个 IP 地址同时只有一个连接。

当指定多个 `limit_conn` 指令时, 任何配置的限制都会适用。

这些指令仅在当前级别未定义任何 `limit_conn` 指令时, 从上一级配置继承。

### limit\_conn\_dry\_run

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>limit_conn_dry_run on   off;</code> |
| 默认值 | <code>limit_conn_dry_run off;</code>      |
| 上下文 | stream, server                            |

启用试运行模式。在此模式下, 连接数并不限于限制, 但在共享内存区域中, 超额连接数的计算照常进行。

### limit\_conn\_log\_level

|     |                                                                 |
|-----|-----------------------------------------------------------------|
| 语法  | <code>limit_conn_log_level info   notice   warn   error;</code> |
| 默认值 | <code>limit_conn_log_level error;</code>                        |
| 上下文 | stream, server                                                  |

设置当服务器限制连接数时的所需日志记录级别。

### limit\_conn\_zone

|     |                                                    |
|-----|----------------------------------------------------|
| 语法  | <code>limit_conn_zone key zone = name:size;</code> |
| 默认值 | —                                                  |
| 上下文 | stream                                             |

设置用于保存各种键状态的共享内存区域的参数。特别是, 状态包含当前连接数。键可以包含文本、变量及其组合。连接的键值为空时不计算在内。

使用示例:



```
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

在此, 由 `$binary_remote_addr` 变量设置的客户端 IP 地址。

对于 IPv4 地址, `$binary_remote_addr` 的大小为 4 字节; 对于 IPv6 地址, 大小为 16 字节。在 32 位平台上, 存储的状态始终占据 32 或 64 字节, 在 64 位平台上则为 64 字节。

一个兆字节的区域可以存储大约 32,000 个 32 字节状态或大约 16,000 个 64 字节状态。如果区域存储被耗尽, 服务器将关闭连接。

## 内置变量

`$limit_conn_status`

保存限制连接数的结果: PASSED, REJECTED 或 REJECTED\_DRY\_RUN

日志 ###

该模块以指定的格式写入请求日志。

## 配置示例

```
log_format basic '$remote_addr [$time_local] '
                '$protocol $status $bytes_sent $bytes_received '
                '$session_time';

access_log /spool/logs/angie-access.log basic buffer=32k;
```

## 指令

### access\_log

|     |                                                                                                                                 |
|-----|---------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>access_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]];</code><br><code>access_log off;</code> |
| 默认  | <code>access_log off;</code>                                                                                                    |
| 上下文 | stream, server                                                                                                                  |

设置用于缓冲日志写入的 `path`、`format` 和配置。在同一级配置中可以指定多个日志。通过在第一个参数中指定前缀 "syslog:" 可以配置日志记录到 `syslog`。特殊值 `off` 会取消当前级别的所有 `access_log` 指令。

如果使用了 `buffer` 或 `gzip` 参数, 日志写入将被缓冲。

**小心**

缓冲区大小不得超过对磁盘文件的原子写入大小。在 FreeBSD 上, 此大小是无限制的。

启用缓冲时, 数据将会写入文件:

- 如果下一行日志无法放入缓冲区;
- 如果缓冲数据的时间超过了 `flush` 参数指定的时间;
- 当工作进程重新打开日志文件 或关闭时。

如果使用了 `gzip` 参数, 那么缓冲数据在写入文件前将被压缩。压缩级别可以设置为 `1` (最快, 压缩最小) 到 `9` (最慢, 压缩最佳) 之间。默认情况下, 缓冲区大小为 `64K` 字节, 压缩级别设为 `1`。由于数据以原子块形式压缩, 日志文件可以在任何时候通过 `zcat` 解压或读取。

示例:

```
access_log /path/to/log.gz basic gzip flush=5m;
```

**重要**

为了使 `gzip` 压缩正常工作, Angie 必须与 `zlib` 库一起构建。

文件路径可以包含变量, 但这样的日志有一些限制:

- 使用工作进程凭据的用户 应有权限在记录日志的目录中创建文件;
- 缓冲写入不起作用;
- 每次写入日志时文件都会被打开和关闭。不过, 由于经常使用的文件描述符可以存储在缓存中, 写入旧文件可以在由 `open_log_file_cache` 指令的 `valid` 参数指定的时间内继续。

`if` 参数启用条件日志记录。如果条件为 `"0"` 或空字符串, 则会话不会被记录。

**log\_format**

|     |                                                                         |
|-----|-------------------------------------------------------------------------|
| 语法  | <code>log_format name [escape=default   json   none] string ...;</code> |
| 默认  | —                                                                       |
| 上下文 | <code>stream, server</code>                                             |

指定日志格式。

`escape` 参数允许在变量中设置 `json` 或 `default` 字符转义, 默认情况下使用 `default` 转义。值 `none` 禁用转义。

对于 `default` 转义, 字符””、”\” 以及值小于 32 或大于 126 的其他字符将被转义为”\xXX”。如果未找到变量值, 将记录为连字符”-”。

对于 `json` 转义, 所有不允许在 JSON 字符串中出现的字符将被转义: 字符””” 和”\” 被转义为”\”” 和”\\”, 值小于 32 的字符被转义为”\n”、”\r”、”\t”、”\b”、”\f” 或”\u00XX”。

### open\_log\_file\_cache

|     |                                                                                     |
|-----|-------------------------------------------------------------------------------------|
| 语法  | <code>open_log_file_cache max = N [inactive=time] [min_uses=N] [valid=time];</code> |
| 默认  | <code>open_log_file_cache off;</code>                                               |
| 上下文 | stream, server                                                                      |

定义一个缓存, 用于存储包含变量名称的日志文件的文件描述符。该指令具有以下参数:

|                       |                                                                     |
|-----------------------|---------------------------------------------------------------------|
| <code>max</code>      | 设置缓存中的最大描述符数量; 如果缓存已满, 则最少最近使用 (LRU) 的描述符将被关闭。                      |
| <code>inactive</code> | 设置缓存描述符在无访问期间关闭的时间; 默认情况下为 10 秒。                                    |
| <code>min_uses</code> | 设置在 <code>inactive</code> 参数定义的时间段内文件使用的最小次数, 以使描述符留在缓存中; 默认情况下为 1。 |
| <code>valid</code>    | 设置检查文件是否仍以相同名称存在的时间; 默认情况下为 60 秒。                                   |
| <code>off</code>      | 禁用缓存。                                                               |

使用示例:

```
open_log_file_cache max=1000 inactive=20s valid=1m min_uses=2;
```

映射 ###

该模块创建的变量, 其值依赖于其他变量的值。

### 配置示例

```
map $remote_addr $limit {
    127.0.0.1    "";
    default     $binary_remote_addr;
}

limit_conn_zone $limit zone=addr:10m;
limit_conn addr 1;
```

## 指令

### map

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>map string \$variable { ... };</code> |
| 默认  | —                                           |
| 上下文 | stream                                      |

创建一个新变量。其值取决于第一个参数，该参数作为包含变量的字符串来指定，例如：

```
set $var1 "foo";
set $var2 "bar";

map $var1$var2 $new_variable {
    default "foobar_value";
}
```

在这里，变量 `$new_variable` 的值将由两个变量 `$var1` 和 `$var2` 组合而成，如果这些变量未定义，则使用默认值。

#### **i** 备注

由于变量仅在使用时才会被评估，即使声明了大量的“map”变量，也不会对请求处理增加任何额外的开销。

在 `map` 块内的参数指定了源值和结果值之间的映射。

源值可以是字符串或正则表达式。

字符串匹配时忽略大小写。

正则表达式应以“~”符号开头以进行区分大小写匹配，或以“~\*”符号开头以进行不区分大小写匹配。正则表达式可以包含命名捕获和位置捕获，这些捕获可以在其他指令中与结果变量一起使用。

如果源值与下述特殊参数的名称之一匹配，则应在其前面加上“”符号。

结果值可以包含文本、变量及其组合。

下列特殊参数也支持：

|                            |                                                               |
|----------------------------|---------------------------------------------------------------|
| <code>default value</code> | 如果源值不匹配指定的任何变体，则设置结果值。当未指定 <code>default</code> 时，默认结果值为空字符串。 |
| <code>hostnames</code>     | 表示源值可以是带有前缀或后缀掩码的主机名。该参数应在值列表之前指定。                            |

例如，

```
*.example.com 1;  
example.*      1;
```

以下两条记录

```
example.com    1;  
*.example.com 1;
```

可以合并为:

```
.example.com 1;
```

|                           |                     |
|---------------------------|---------------------|
| <code>include file</code> | 包含一个带有值的文件。可以有多个包含。 |
| <code>volatile</code>     | 表示变量不可缓存。           |

如果源值匹配多个指定的变体, 例如同时匹配掩码和正则表达式, 则会选择第一个匹配的变体, 优先顺序如下:

1. 没有掩码的字符串值
2. 具有前缀掩码的最长字符串值, 例如”\*.example.com”
3. 具有后缀掩码的最长字符串值, 例如”mail.\*”
4. 第一个匹配的正则表达式 (按照配置文件中的出现顺序)
5. 默认值 (*default*)

### map\_hash\_bucket\_size

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>map_hash_bucket_size size;</code>      |
| 默认  | <code>map_hash_bucket_size 32 64 128;</code> |
| 上下文 | stream                                       |

设置 `map` 变量哈希表的桶大小。默认值取决于处理器的缓存行大小。有关设置哈希表的详细信息, 请参阅 [单独的文档](#)。

## map\_hash\_max\_size

|     |                         |
|-----|-------------------------|
| 语法  | map_hash_max_size size; |
| 默认  | map_hash_max_size 2048; |
| 上下文 | stream                  |

设置 `map` 变量哈希表的最大大小。有关设置哈希表的详细信息, 请参阅 [单独的文档](#)。

## MQTT 预读

启用从消息队列遥测传输 (MQTT) 版本的 `CONNECT` 数据包中提取客户端 ID 和用户名 3.1.1 和 5.0。

当从源代码 构建时, 该模块默认不被构建; 应通过 `--with-stream_mqtt_preread_module` 构建选项启用。

在来自 我们的仓库的软件包和镜像中, 该模块已包含在构建中。

## 配置示例

### 通过客户端 ID 选择上游服务器:

```
stream {  
  
    mqtt_preread on;  
  
    upstream mqtt {  
        hash $mqtt_preread_clientid;  
        # ...  
    }  
}
```

## 指令

### mqtt\_preread

|     |                        |
|-----|------------------------|
| 语法  | mqtt_preread on   off; |
| 默认  | mqtt_preread off;      |
| 上下文 | stream, server         |

控制在预读阶段 从 `CONNECT` 数据包中提取信息。如果设置为 `on`, 其周围的上下文将填充以下变量。

## 内置变量

查看 MQTT 版本 3.1.1 和 5.0 规范中的值语义详情。

`$mqtt_preread_clientid`

唯一客户端 ID。

`$mqtt_preread_username`

可选用户名。

## 传递

允许将已接受的连接直接传递到在 *HTTP*、*Stream* 或 *Mail* 模块中配置的任何监听套接字。

## 配置示例

在 `stream` 模块处理 SSL/TLS 终止后, 连接被转发到 `http` 模块:

```
http {
    server {
        listen 8000;

        location / {
            root html;
        }
    }
}

stream {
    server {
        listen 12345 ssl;

        ssl_certificate     domain.crt;
        ssl_certificate_key domain.key;

        pass 127.0.0.1:8000;
    }
}
```

## 指令

### pass

|         |                            |
|---------|----------------------------|
| Syntax  | <code>pass address;</code> |
| 默认      | —                          |
| Context | server                     |

该指令设置客户端连接应传递到的服务器地址。 *address* 可以作为 IP 地址和端口给出:

```
pass 127.0.0.1:12345;
```

或者作为 UNIX 域套接字的路径:

```
pass unix:/tmp/stream.socket;
```

此外, *address* 也可以用变量设置:

```
pass $upstream;
```

## 代理

允许通过 TCP、UDP 和 UNIX 域套接字代理数据流。

## 配置示例

```
server {
    listen 127.0.0.1:12345;
    proxy_pass 127.0.0.1:8080;
}

server {
    listen 12345;
    proxy_connect_timeout 1s;
    proxy_timeout 1m;
    proxy_pass example.com:12345;
}

server {
    listen 53 udp reuseport;
    proxy_timeout 20s;
    proxy_pass dns.example.com:53;
}
```



```
server {  
    listen [::]:12345;  
    proxy_pass unix:/tmp/stream.socket;  
}
```

## 指令

### proxy\_bind

|     |                                                      |
|-----|------------------------------------------------------|
| 语法  | <code>proxy_bind address [transparent]   off;</code> |
| 默认  | —                                                    |
| 上下文 | stream, server                                       |

使到代理服务器的出站连接来源于指定的本地 IP 地址。参数值可以包含变量。特殊值 `off` 取消从先前配置级别继承的 `proxy_bind` 指令的效果, 允许系统自动分配本地 IP 地址。

`transparent` 参数允许到代理服务器的出站连接来源于非本地 IP 地址, 例如, 来自客户端的真实 IP 地址:

```
proxy_bind $remote_addr transparent;
```

要使此参数生效, Angie 工作进程通常需要以超级用户 权限运行。在 Linux 上, 若指定 `transparent` 参数, 则工作进程从主进程继承 `CAP_NET_RAW` 能力, 这不是必需的。

#### ❗ 重要

内核路由表也应配置以拦截来自 FastCGI 服务器的网络流量。

### proxy\_buffer\_size

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>proxy_buffer_size size;</code> |
| 默认  | <code>proxy_buffer_size 16k;</code>  |
| 上下文 | stream, server                       |

设置用于从代理服务器读取数据的缓冲区大小。也设置从客户端读取数据的缓冲区大小。

### proxy\_connect\_timeout

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>proxy_connect_timeout time;</code> |
| 默认  | <code>proxy_connect_timeout 60s;</code>  |
| 上下文 | stream, server                           |

定义与代理服务器建立连接的超时时间。

### proxy\_connection\_drop

|     |                                                     |
|-----|-----------------------------------------------------|
| 语法  | <code>proxy_connection_drop time   on   off;</code> |
| 默认  | <code>proxy_connection_drop off;</code>             |
| 上下文 | stream, server                                      |

启用在代理服务器被从组中移除或被 *eresolve* 进程或 *API* 命令 `DELETE` 标记为永久不可用后终止与其中的所有会话。

在为客户端或代理服务器处理下一个读取或写入事件时, 终止会话。

设置 *time* 启用会话终止 超时; 设置 `on` 时, 会话立即终止。

### proxy\_download\_rate

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>proxy_download_rate rate;</code> |
| 默认  | <code>proxy_download_rate 0;</code>    |
| 上下文 | stream, server                         |

限制从代理服务器读取数据的速度。rate 以每秒字节数指定。

|   |        |
|---|--------|
| 0 | 禁用速率限制 |
|---|--------|

#### **i** 备注

限制是针对每个连接设置的, 因此如果 *Angie* 同时打开两个到代理服务器的连接, 总速率将是指定限制的两倍。

参数值可以包含变量。在速率应根据某个条件限制的情况下可能会有用:

```
proxy_download_rate $rate;
```

```
map $slow $rate {  
    1    4k;  
    2    8k;  
}
```

### proxy\_half\_close

|     |                            |
|-----|----------------------------|
| 语法  | proxy_half_close on   off; |
| 默认  | proxy_half_close off;      |
| 上下文 | stream, server             |

启用或禁用独立关闭 TCP 连接的每个方向 (“TCP 半关闭”)。如果启用, TCP 上的代理将在双方关闭连接之前保持。

### proxy\_next\_upstream

|     |                               |
|-----|-------------------------------|
| 语法  | proxy_next_upstream on   off; |
| 默认  | proxy_next_upstream on;       |
| 上下文 | stream, server                |

当无法建立与代理服务器的连接时, 确定是否将客户端连接传递给下一个上游池中的服务器。

传递连接给下一个服务器可以通过尝试次数 和时间 来限制。

### proxy\_next\_upstream\_timeout

|     |                                           |
|-----|-------------------------------------------|
| 语法  | proxy_next_upstream_timeout <i>time</i> ; |
| 默认  | proxy_next_upstream_timeout 0;            |
| 上下文 | stream, server                            |

限制传递连接到下一个服务器的允许时间。

|   |       |
|---|-------|
| 0 | 关闭此限制 |
|---|-------|

### proxy\_next\_upstream\_tries

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>proxy_next_upstream_tries number;</code> |
| 默认  | <code>proxy_next_upstream_tries 0;</code>      |
| 上下文 | stream, server                                 |

限制尝试将连接传递到下一个服务器的可能次数。

|   |       |
|---|-------|
| 0 | 关闭此限制 |
|---|-------|

### proxy\_pass

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>proxy_pass address;</code> |
| 默认  | —                                |
| 上下文 | server                           |

设置代理服务器的地址。address 可以指定为域名或 IP 地址和端口：

```
proxy_pass localhost:12345;
```

或为 UNIX 域套接字路径：

```
proxy_pass unix:/tmp/stream.socket;
```

如果域名解析为多个地址，则会以循环方式使用所有这些地址。此外，地址可以指定为服务器组。如果使用了组，则不能与其一起指定端口；而应分别为组内的每个服务器指定端口。

地址还可以使用变量指定：

```
proxy_pass $upstream;
```

在这种情况下，在描述的服务器组中查找服务器名称，如果未找到，则使用resolver 确定。

### proxy\_protocol

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>proxy_protocol on   off;</code> |
| 默认  | <code>proxy_protocol off;</code>      |
| 上下文 | stream, server                        |

为连接到代理服务器启用 PROXY 协议。

### proxy\_requests

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>proxy_requests number;</code> |
| 默认  | <code>proxy_requests 0;</code>      |
| 上下文 | stream, server                      |

设置在客户端与现有 UDP 流会话之间解除绑定时的客户端数据报数量。在接收到指定数量的数据报后，同一客户端的下一个数据报开始一个新会话。当所有客户端数据报都传输到代理服务器并接收到预期的响应数量时，或在达到超时时，会话终止。

### proxy\_responses

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>proxy_responses number;</code> |
| 默认  | —                                    |
| 上下文 | stream, server                       |

设置在使用 UDP 协议时期望代理服务器响应的客户端数据报数量。该数量作为会话终止的提示。默认情况下，数据报数量不受限制。

如果指定零值，则不期望响应。然而，如果接收到响应且会话仍未完成，则将处理该响应。

### proxy\_socket\_keepalive

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>proxy_socket_keepalive on   off;</code> |
| 默认  | <code>proxy_socket_keepalive off;</code>      |
| 上下文 | stream, server                                |

配置到代理服务器的出站连接的“TCP keepalive”行为。

|    |                            |
|----|----------------------------|
| "" | 默认情况下，套接字的操作系统设置生效。        |
| on | 为套接字打开 SO_KEEPALIVE 套接字选项。 |

### proxy\_ssl

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>proxy_ssl on   off;</code> |
| 默认  | <code>proxy_ssl off;</code>      |
| 上下文 | stream, server                   |

为连接到代理服务器启用 SSL/TLS 协议。

### proxy\_ssl\_certificate

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>proxy_ssl_certificate file;</code> |
| 默认  | —                                        |
| 上下文 | stream, server                           |

指定用于对代理服务器进行身份验证的 PEM 格式证书文件。文件名中可以使用变量。

Added in version 1.2.0.

当启用 `proxy_ssl_ntls` 时, 该指令接受两个参数而非一个, 即证书的签名和加密部分:

```
server {
    proxy_ssl_ntls on;

    proxy_ssl_certificate    sign.crt enc.crt;
    proxy_ssl_certificate_key sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass backend:12345;
}
```

### proxy\_ssl\_certificate\_key

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>proxy_ssl_certificate_key file;</code> |
| 默认值 | —                                            |
| 上下文 | stream, server                               |

指定用于验证代理服务器的 PEM 格式的秘密密钥文件。变量可以在文件名中使用。

Added in version 1.2.0.

当启用 `proxy_ssl_ntls` 时, 该指令接受两个参数而非一个, 即密钥的签名和加密部分:

```
server {
    proxy_ssl_ntls on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass backend:12345;
}
```

### proxy\_ssl\_ciphers

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>proxy_ssl_ciphers <i>ciphers</i>;</code> |
| 默认值 | <code>proxy_ssl_ciphers DEFAULT;</code>        |
| 上下文 | stream, server                                 |

指定请求代理服务器时启用的加密套件。加密套件以 OpenSSL 库理解的格式指定。

完整列表可通过“`openssl ciphers`”命令查看。

### proxy\_ssl\_conf\_command

|     |                                                        |
|-----|--------------------------------------------------------|
| 语法  | <code>proxy_ssl_conf_command <i>name value</i>;</code> |
| 默认值 | —                                                      |
| 上下文 | stream, server                                         |

在与代理服务器建立连接时，设置任意 OpenSSL 配置命令。

#### 重要

该指令在使用 OpenSSL 1.0.2 或更高版本时支持。

多个 `proxy_ssl_conf_command` 指令可以在同一级别指定。只有在当前级别没有定义 `proxy_ssl_conf_command` 指令时，这些指令才会从先前的配置级别继承。

#### 小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

### proxy\_ssl\_crl

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>proxy_ssl_crl file;</code> |
| 默认值 | —                                |
| 上下文 | stream, server                   |

指定用于验证代理服务器证书的 PEM 格式的撤销证书文件 (CRL)。

### proxy\_ssl\_name

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>proxy_ssl_name name;</code>         |
| 默认值 | <code>proxy_ssl_name \$proxy_host;</code> |
| 上下文 | stream, server                            |

允许覆盖用于验证代理服务器证书的服务器名称, 并在与代理服务器建立连接时通过 SNI 传递。

默认情况下, 使用 `proxy_pass` 地址的主机部分。

### proxy\_ssl\_ntls

Added in version 1.2.0.

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>proxy_ssl_ntls on   off;</code> |
| 默认值 | <code>proxy_ssl_ntls off;</code>      |
| 上下文 | stream, server                        |

启用使用 `TongSuo` 库的客户端 NTLS 支持。

```
server {
    proxy_ssl_ntls on;

    proxy_ssl_certificate      sign.crt enc.crt;
    proxy_ssl_certificate_key  sign.key enc.key;

    proxy_ssl_ciphers "ECC-SM2-WITH-SM4-SM3:ECDHE-SM2-WITH-SM4-SM3:RSA";

    proxy_pass backend:12345;
}
```



**重要**

使用 `--with-ntls` 构建选项构建 Angie, 并链接具有 NTLS 支持的 SSL 库

```
./configure --with-openssl=../Tongsuo-8.3.0 \  
            --with-openssl-opt=enable-ntls \  
            --with-ntls
```

**proxy\_ssl\_password\_file**

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>proxy_ssl_password_file file;</code> |
| 默认值 | —                                          |
| 上下文 | stream, server                             |

指定用于秘密密钥的密码文件, 其中每个密码单独占一行。在加载密钥时依次尝试这些密码。

**proxy\_ssl\_protocols**

|     |                                                                                         |
|-----|-----------------------------------------------------------------------------------------|
| 语法  | <code>proxy_ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];</code> |
| 默认值 | <code>proxy_ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;</code>                         |
| 上下文 | stream, server                                                                          |

在 1.2.0 版本发生变更: 默认集合中添加了 TLSv1.3 参数。

启用请求代理服务器时使用的指定协议。

**proxy\_ssl\_server\_name**

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>proxy_ssl_server_name on   off;</code> |
| 默认值 | <code>proxy_ssl_server_name off;</code>      |
| 上下文 | stream, server                               |

启用或禁用通过 `proxy_ssl_name` 指令设置的服务器名称, 通过 `服务器名称指示` TLS 扩展 (SNI, RFC 6066) 在与代理服务器建立连接时传递。

### proxy\_ssl\_session\_reuse

|     |                                                |
|-----|------------------------------------------------|
| 语法  | <code>proxy_ssl_session_reuse on   off;</code> |
| 默认值 | <code>proxy_ssl_session_reuse on;</code>       |
| 上下文 | stream, server                                 |

决定在与代理服务器交互时是否可以重用 SSL 会话。如果日志中出现“*SSL3\_GET\_FINISHED:digest check failed*”错误, 请尝试禁用会话重用。

### proxy\_ssl\_trusted\_certificate

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>proxy_ssl_trusted_certificate file;</code> |
| 默认值 | —                                                |
| 上下文 | http, server, location                           |

指定用于验证代理服务器证书的 PEM 格式的受信任 CA 证书文件。

### proxy\_ssl\_verify

|     |                                         |
|-----|-----------------------------------------|
| 语法  | <code>proxy_ssl_verify on   off;</code> |
| 默认值 | <code>proxy_ssl_verify off;</code>      |
| 上下文 | stream, server                          |

启用或禁用对代理服务器证书的验证。

### proxy\_ssl\_verify\_depth

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>proxy_ssl_verify_depth number;</code> |
| 默认值 | <code>proxy_ssl_verify_depth 1;</code>      |
| 上下文 | stream, server                              |

设置代理服务器证书链的验证深度。

## proxy\_timeout

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>proxy_timeout timeout;</code> |
| 默认值 | <code>proxy_timeout 10m;</code>     |
| 上下文 | stream, server                      |

设置客户端或代理服务器连接上两次连续读写操作之间的超时时间。如果在此时间内没有传输数据, 则关闭连接。

## upstream\_probe\_timeout (PRO)

Added in version 1.4.0: PRO

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>upstream_probe_timeout time;</code> |
| 默认值 | <code>upstream_probe_timeout 50s;</code>  |
| 上下文 | server                                    |

设置通过 `upstream_probe (PRO)` 指令配置的探测的已建立服务器连接的最大不活动时间; 如果超过此限制, 连接将被关闭。

## proxy\_upload\_rate

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>proxy_upload_rate rate;</code> |
| 默认值 | <code>proxy_upload_rate 0;</code>    |
| 上下文 | stream, server                       |

限制从客户端读取数据的速度。rate 以每秒字节数指定。

|   |        |
|---|--------|
| 0 | 禁用速率限制 |
|---|--------|

### 备注

该限制是针对每个连接设置的, 因此如果客户端同时打开两个连接, 总速率将是指定限制的两倍。

参数值可以包含变量。在需要根据某些条件限制速率的情况下可能会有用:

```
map $slow $rate {  
    1    4k;  
    2    8k;
```

```
}  
  
proxy_upload_rate $rate;
```

## RDP 预读

在使用 RDP 协议时, 此模块允许在做出负载均衡决策之前提取用于会话识别和管理的 Cookie。

当从源代码构建时, 此模块默认不构建; 需要通过 `--with-stream_rdp_preread_module` 构建选项来启用。

在我们的仓库中的包和镜像中, 该模块已包含在构建中。

## 配置示例

### 绑定到发放 Cookie 的服务器

这使用了 `sticky` 指令的 `learn` 模式:

```
stream {  
  
    rdp_preread on;  
  
    upstream rdp {  
  
        server 127.0.0.1:3390 sid=a;  
        server 127.0.0.1:3391 sid=b;  
  
        sticky learn lookup=$rdp_cookie create=$rdp_cookie zone=sessions:1m;  
    }  
}
```

## 指令

### rdp\_preread

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>rdp_preread on   off;</code> |
| 默认值 | <code>rdp_preread off;</code>      |
| 上下文 | <code>stream, server</code>        |

控制在预读阶段从 RDP 协议 Cookie 中提取信息。如果设置为 `on`, 其周围的上下文将会填充以下变量。

## 内置变量

Cookie 值的语义取决于 RDP 协议版本。

`$rdp_cookie`

整个 Cookie 值。

`$rdp_cookie_<name>`

具有指定 *name* 的 Cookie 字段的值。

## RealIP

该模块用于将客户端地址和端口更改为在 PROXY 协议头中发送的地址和端口。必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

当从源代码构建时，默认情况下不会构建此模块；应使用 `--with-stream_realip_module` 构建选项来启用它。

在我们的仓库中的包和镜像中，该模块已包含在构建中。

## 配置示例

```
listen 12345 proxy_protocol;

set_real_ip_from 192.168.1.0/24;
set_real_ip_from 192.168.2.1;
set_real_ip_from 2001:0db8::/32;
```

## 指令

`set_real_ip_from`

|     |                                                        |
|-----|--------------------------------------------------------|
| 语法  | <code>set_real_ip_from address   CIDR   unix: ;</code> |
| 默认值 | —                                                      |
| 上下文 | stream, server                                         |

定义已知发送正确替换地址的可信地址。如果指定了特殊值 `unix:`，则所有 UNIX 域套接字将被信任。

## 内置变量

`$realip_remote_addr`

保留原始客户端地址

`$realip_remote_port`

保留原始客户端端口

## 返回

该模块允许向客户端发送指定的值，然后关闭连接。

## 配置示例

```
server {  
    listen 12345;  
    return $time_iso8601;  
}
```

## 指令

### return

|     |                            |
|-----|----------------------------|
| 语法  | <code>return value;</code> |
| 默认值 | —                          |
| 上下文 | server                     |

指定要发送给客户端的值。该值可以包含文本、变量及其组合。

设置 ###

该模块允许为变量设置一个值。

## 配置示例

```
server {  
    listen 12345;  
    set    $true 1;  
}
```

## 指令

### set

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>set \$variable value;</code> |
| 默认值 | —                                  |
| 上下文 | server                             |

为指定的变量设置一个值。该值可以包含文本、变量及其组合。

## 拆分客户端

该模块用于 A/B 测试、金丝雀发布或其他需要将一定比例的客户端路由到一个服务器或配置的场景，同时将剩余部分引导到其他地方。

## 配置示例

```
stream {  
    # ...  
    split_clients "${remote_addr}AAA" $upstream {  
        0.5%         feature_test1;  
        2.0%         feature_test2;  
        *            production;  
    }  
  
    server {  
        # ...  
        proxy_pass $upstream;  
    }  
}
```

## 指令

### split\_clients

|         |                                                      |
|---------|------------------------------------------------------|
| Syntax  | <code>split_clients string \$variable { ... }</code> |
| 默认值     | —                                                    |
| Context | stream                                               |

通过对 *string* 进行哈希来创建一个 *\$variable*; *string* 中的变量被替换, 结果被哈希, 然后哈希被映射到 *\$variable* 的字符串值。

哈希函数使用 `MurmurHash2` (32 位), 其整个值范围 (0 到 4294967295) 按出现顺序映射到桶; 百分比决定了桶的大小。通配符 (\*) 可以出现在最后; 落在其他桶外的哈希被映射到其分配的值。

示例:

```
split_clients "${remote_addr}AAA" $variant {  
    0.5%           .one;  
    2.0%           .two;  
    *              "";  
}
```

这里, 插值后的 `$remote_addrAAA` 字符串的哈希值分布如下:

- 值为 0 到 21474835 (0.5% 的样本) 产生 `.one`
- 值为 21474836 到 107374180 (2% 的样本) 产生 `.two`
- 值为 107374181 到 4294967295 (所有其他值) 产生 `""` (一个空字符串)

## SSL

提供流代理服务器与 SSL/TLS 协议一起工作的必要支持。

当从源码 构建时, 此模块默认未构建; 应该使用 `--with-stream_ssl_module` 构建选项启用。

在来自 我们的仓库的包和镜像中, 模块已包含在构建中。

### 重要

此模块需要 OpenSSL 库。



## 配置示例

为了减少处理器负载, 建议

- 将工作进程 的数量设置为等于处理器数量,
- 启用共享 会话缓存,
- 禁用内置 会话缓存,
- 并可能增加会话生命周期 (默认 5 分钟):

```
worker_processes auto;

stream {

    #...

    server {
        listen          12345 ssl;

        ssl_protocols   TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
        ssl_ciphers      AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate   /usr/local/angie/conf/cert.pem;
        ssl_certificate_key /usr/local/angie/conf/cert.key;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;

        # ...
    }
}
```

## 指令

### ssl\_alpn

|     |                        |
|-----|------------------------|
| 语法  | ssl_alpn protocol ...; |
| 默认值 | —                      |
| 上下文 | stream, server         |

指定支持的 ALPN 协议列表。如果客户端使用 ALPN, 则必须协商 其中一个协议:

```
map $ssl_alpn_protocol $proxy {
    h2          127.0.0.1:8001;
    http/1.1    127.0.0.1:8002;
}

server {
```

```
listen      12346;  
proxy_pass $proxy;  
ssl_alpn    h2 http/1.1;  
}
```

## ssl\_certificate

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>ssl_certificate file [file];</code> |
| 默认值 | —                                         |
| 上下文 | stream, server                            |

指定给定服务器的 PEM 格式证书文件。如果需要指定中间证书, 应在同一文件中按以下顺序指定: 首先是主证书, 然后是中间证书。可以将 PEM 格式的私钥放在同一文件中。

此指令可以多次指定以加载不同类型的证书, 例如 RSA 和 ECDSA:

```
server {  
    listen          12345 ssl;  
  
    ssl_certificate    example.com.rsa.crt;  
    ssl_certificate_key example.com.rsa.key;  
  
    ssl_certificate    example.com.ecdsa.crt;  
    ssl_certificate_key example.com.ecdsa.key;  
  
    # ...  
}
```

只有 OpenSSL 1.0.2 或更高版本支持为不同证书提供单独的证书链。在较旧版本中, 只能使用一个证书链。

### 重要

使用 OpenSSL 1.0.2 或更高版本时, 可以在文件名中使用变量:

```
ssl_certificate    $ssl_server_name.crt;  
ssl_certificate_key $ssl_server_name.key;
```

请注意, 使用变量意味着每次 SSL 握手都会加载证书, 这可能会对性能产生负面影响。

可以指定 “data:\$variable” 作为 file 的值, 从而从变量中加载证书而无需使用中间文件。

请注意, 不当使用此语法可能会产生安全隐患, 例如将秘密密钥数据写入错误日志。

Added in version 1.2.0.

当启用 `ssl_ntls` 时, 指令接受两个参数而不是一个: 证书的签名和加密部分:

```
listen ... ssl;

ssl_ntls on;

# 双 NTLS 证书
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规 RSA 证书结合使用:
ssl_certificate      rsa.crt;
ssl_certificate_key  rsa.key;
```

### ssl\_certificate\_key

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>ssl_certificate_key file [file];</code> |
| 默认值 | —                                             |
| 上下文 | stream, server                                |

指定给定服务器的 PEM 格式私钥文件。

#### 重要

使用 OpenSSL 1.0.2 或更高版本时, 可以在文件名中使用变量。

可以指定 “engine:name:id” 作为 `file` 的值, 从 OpenSSL 引擎 `name` 中加载具有指定 `id` 的私钥。

可以指定 “data:\$variable” 作为 `file` 的值, 从变量中加载私钥而无需使用中间文件。请注意, 不当使用此语法可能会产生安全隐患, 例如将秘密密钥数据写入错误日志。

Added in version 1.2.0.

当启用 `ssl_ntls` 时, 指令接受两个参数而不是一个: 密钥的签名和加密部分:

```
listen ... ssl;

ssl_ntls on;

# 双 NTLS 证书
ssl_certificate      sign.crt enc.crt;
ssl_certificate_key  sign.key enc.key;

# 可以与常规 RSA 证书结合使用:
ssl_certificate      rsa.crt;
ssl_certificate_key  rsa.key;
```

## ssl\_ciphers

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>ssl_ciphers <i>ciphers</i>;</code>   |
| 默认值 | <code>ssl_ciphers HIGH:!aNULL:!MD5;</code> |
| 上下文 | stream, server                             |

指定启用的密码。密码以 OpenSSL 库能理解的格式指定, 例如:

```
ssl_ciphers ALL:!aNULL:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

可以使用 “`openssl ciphers`” 命令查看完整列表。

## ssl\_client\_certificate

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>ssl_client_certificate <i>file</i>;</code> |
| 默认值 | —                                                |
| 上下文 | stream, server                                   |

指定用于验证 客户端证书的 PEM 格式的受信 CA 证书文件。

证书列表将发送给客户端。如果不希望这样, 可以使用 `ssl_trusted_certificate` 指令。

## ssl\_conf\_command

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>ssl_conf_command <i>name value</i>;</code> |
| 默认值 | —                                                |
| 上下文 | stream, server                                   |

设置任意 OpenSSL 配置 命令。


### 重要

使用 OpenSSL 1.0.2 或更高版本时支持该指令。

可以在同一级别指定多个 `ssl_conf_command` 指令:

```
ssl_conf_command Options PrioritizeChaCha;  
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

如果且仅当当前级别未定义 `ssl_conf_command` 指令时, 这些指令从上一级配置继承。

 小心

请注意, 直接配置 OpenSSL 可能导致意外行为。

### ssl\_crl


|     |                            |
|-----|----------------------------|
| 语法  | <code>ssl_crl file;</code> |
| 默认值 | —                          |
| 上下文 | stream, server             |

指定用于验证 客户端证书的 PEM 格式的吊销证书 (CRL) 文件。

### ssl\_dhparam

|     |                                |
|-----|--------------------------------|
| 语法  | <code>ssl_dhparam file;</code> |
| 默认值 | —                              |
| 上下文 | stream, server                 |

指定用于 DHE 密码的 DH 参数文件。

 小心

默认情况下没有设置参数, 因此不会使用 DHE 密码。

### ssl\_ecdh\_curve

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>ssl_ecdh_curve curve;</code> |
| 默认值 | <code>ssl_ecdh_curve auto;</code>  |
| 上下文 | stream, server                     |

指定用于 ECDHE 密码的曲线。

 重要

使用 OpenSSL 1.0.2 或更高版本时, 可以指定多个曲线, 例如:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

特殊值 `auto` 指示 Angie 使用 OpenSSL 库内置列表（使用 OpenSSL 1.0.2 或更高版本时）或使用旧版本的 `prime256v1`。

### 重要

使用 OpenSSL 1.0.2 或更高版本时，此指令设置服务器支持的曲线列表。因此，为了使 ECDSA 证书正常工作，重要的是包括证书中使用的曲线。

## ssl\_handshake\_timeout

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>ssl_handshake_timeout time;</code> |
| 默认值 | <code>ssl_handshake_timeout 60s;</code>  |
| 上下文 | <code>stream, server</code>              |

指定 SSL 握手完成的超时时间。

## ssl\_ntls

Added in version 1.2.0.

|     |                                 |
|-----|---------------------------------|
| 语法  | <code>ssl_ntls on   off;</code> |
| 默认值 | <code>ssl_ntls off;</code>      |
| 上下文 | <code>stream, server</code>     |

启用使用 TongSuo 库的 NTLS 服务端支持。

```
listen ... ssl;  
ssl_ntls on;
```

### 重要

使用 `--with-ntls` 构建选项构建 Angie 并与支持 NTLS 的 SSL 库链接

```
./configure --with-openssl=../Tongsuo-8.3.0 \  
            --with-openssl-opt=enable-ntls \  
            --with-ntls
```

## ssl\_password\_file

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>ssl_password_file file;</code> |
| 默认值 | —                                    |
| 上下文 | stream, server                       |

指定用于秘密密钥 的密码的文件, 其中每个密码单独一行。当加载密钥时, 依次尝试密码。

示例:

```
stream {
    ssl_password_file /etc/keys/global.pass;
    ...

    server {
        listen 127.0.0.1:12345;
        ssl_certificate_key /etc/keys/first.key;
    }

    server {
        listen 127.0.0.1:12346;

        # 也可以使用命名管道代替文件
        ssl_password_file /etc/keys/fifo;
        ssl_certificate_key /etc/keys/second.key;
    }
}
```

## ssl\_prefer\_server\_ciphers

|     |                                                  |
|-----|--------------------------------------------------|
| 语法  | <code>ssl_prefer_server_ciphers on   off;</code> |
| 默认值 | <code>ssl_prefer_server_ciphers off;</code>      |
| 上下文 | stream, server                                   |

指定在使用 SSLv3 和 TLS 协议时, 服务器密码应优先于客户端密码。

## ssl\_protocols

|     |                                                                                   |
|-----|-----------------------------------------------------------------------------------|
| 语法  | <code>ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3];</code> |
| 默认值 | <code>ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;</code>                         |
| 上下文 | <code>stream, server</code>                                                       |

在 1.2.0 版本发生变更: TLSv1.3 参数已添加到默认集。

启用指定的协议。

### 重要

TLSv1.1 和 TLSv1.2 参数仅在使用 OpenSSL 1.0.1 或更高版本时有效。

TLSv1.3 参数仅在使用 OpenSSL 1.1.1 或更高版本时有效。

## ssl\_session\_cache

|     |                                                                                  |
|-----|----------------------------------------------------------------------------------|
| 语法  | <code>ssl_session_cache off   none   [builtin[:size]] [shared:name:size];</code> |
| 默认值 | <code>ssl_session_cache none;</code>                                             |
| 上下文 | <code>stream, server</code>                                                      |

设置存储会话参数的缓存类型和大小。缓存可以是以下类型之一：

|                      |                                                                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>off</code>     | 严格禁止使用会话缓存：Angie 明确告知客户端会话不可重用。                                                                                                                                |
| <code>none</code>    | 温和地不允许使用会话缓存：Angie 告知客户端会话可以重用，但实际上并不在缓存中存储会话参数。                                                                                                               |
| <code>builtin</code> | 内建于 OpenSSL 的缓存，仅由一个工作进程使用。缓存大小以会话数表示。如果未给出大小，则等于 20480 个会话。使用内建缓存可能导致内存碎片化。                                                                                   |
| <code>shared</code>  | 在所有工作进程之间共享的缓存。缓存大小以字节为单位表示；一兆字节可以存储约 4000 个会话。每个共享缓存应有一个任意名称。具有相同名称的缓存可用于多个服务器中。除非使用 <code>ssl_session_ticket_key</code> 指令显式配置，否则还用于自动生成、存储和定期轮换 TLS 会话票据密钥。 |

可以同时使用两种缓存类型，例如：

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

但仅使用共享缓存而不使用内建缓存应该更为高效。



### ssl\_session\_ticket\_key

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>ssl_session_ticket_key file;</code> |
| 默认值 | —                                         |
| 上下文 | stream, server                            |

设置用于加密和解密 TLS 会话票据的密钥文件。如果需要在多个服务器之间共享相同的密钥, 则该指令是必要的。默认情况下, 将使用随机生成的密钥。

如果指定了多个密钥, 则仅使用第一个密钥来加密 TLS 会话票据。这允许配置密钥轮换, 例如:

```
ssl_session_ticket_key current.key;  
ssl_session_ticket_key previous.key;
```

文件必须包含 80 或 48 字节的随机数据, 可以使用以下命令创建:

```
openssl rand 80 > ticket.key
```

根据文件大小, 使用 AES256 (用于 80 字节的密钥) 或 AES128 (用于 48 字节的密钥) 进行加密。

### ssl\_session\_tickets

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>ssl_session_tickets on   off;</code> |
| 默认值 | <code>ssl_session_tickets on;</code>       |
| 上下文 | stream, server                             |

启用或禁用通过 TLS 会话票据 的会话恢复。

### ssl\_session\_timeout

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>ssl_session_timeout time;</code> |
| 默认值 | <code>ssl_session_timeout 5m;</code>   |
| 上下文 | stream, server                         |

指定客户端可以重用会话参数的时间。

### ssl\_trusted\_certificate

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>ssl_trusted_certificate file;</code> |
| 默认值 | —                                          |
| 上下文 | stream, server                             |

指定一个包含受信任 CA 证书的文件, 格式为 PEM, 用于验证客户端证书。

与 `ssl_client_certificate` 设置的证书不同, 这些证书列表不会发送给客户端。

### ssl\_verify\_client

|     |                                                                      |
|-----|----------------------------------------------------------------------|
| 语法  | <code>ssl_verify_client on   off   optional   optional_no_ca;</code> |
| 默认值 | <code>ssl_verify_client off;</code>                                  |
| 上下文 | stream, server                                                       |

启用客户端证书的验证。验证结果存储在 `$ssl_client_verify` 变量中。如果在客户端证书验证期间发生错误或客户端未提供所需的证书, 则连接将关闭。

|                             |                                                          |
|-----------------------------|----------------------------------------------------------|
| <code>optional</code>       | 请求客户端证书并在证书存在时进行验证。                                      |
| <code>optional_no_ca</code> | 请求客户端证书, 但不要求其由受信任的 CA 证书签名。这适用于 Angie 之外的服务执行实际证书验证的情况。 |

### ssl\_verify\_depth

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>ssl_verify_depth number;</code> |
| 默认值 | <code>ssl_verify_depth 1;</code>      |
| 上下文 | stream, server                        |

设置客户端证书链的验证深度。

### 内建变量

`stream_ssl` 模块支持以下变量:

```
$ssl_alpn_protocol
```

返回在 SSL 握手期间通过 ALPN 选择的协议, 否则返回空字符串。

```
$ssl_cipher
```

返回已建立 SSL 连接使用的密码名称。

```
$ssl_ciphers
```

返回客户端支持的密码列表。已知密码按名称列出, 未知密码以十六进制显示, 例如:

```
AES128-SHA:AES256-SHA:0x00ff
```

#### 重要

该变量仅在使用 OpenSSL 1.0.2 或更高版本时完全支持。在较旧版本中, 该变量仅适用于新会话, 并且仅列出已知密码。

```
$ssl_client_cert
```

返回已建立 SSL 连接的客户端证书, 格式为 PEM, 每行 (除第一行外) 前面加制表符。

```
$ssl_client_fingerprint
```

返回已建立 SSL 连接的客户端证书的 SHA1 指纹。

```
$ssl_client_i_dn
```

返回已建立 SSL 连接的客户端证书的“颁发者 DN”字符串, 依据 RFC 2253。

```
$ssl_client_raw_cert
```

返回已建立 SSL 连接的客户端证书, 格式为 PEM。

```
$ssl_client_s_dn
```

返回已建立 SSL 连接的客户端证书的“主题 DN”字符串, 依据 RFC 2253。

```
$ssl_client_serial
```

返回已建立 SSL 连接的客户端证书的序列号。

```
$ssl_client_v_end
```

返回客户端证书的结束日期。

```
$ssl_client_v_remain
```

返回客户端证书到期的天数。

```
$ssl_client_v_start
```

返回客户端证书的开始日期。

```
$ssl_client_verify
```

返回客户端证书验证结果: SUCCESS, "FAILED:reason" 和 NONE 如果未提供证书。

```
$ssl_curve
```

返回用于 SSL 握手密钥交换过程的协商曲线。已知曲线按名称列出, 未知曲线以十六进制显示, 例如:

```
prime256v1
```

#### 重要

该变量仅在使用 OpenSSL 3.0 或更高版本时支持。在较旧版本中, 变量值将为空字符串。

`$ssl_curves`

返回客户端支持的曲线列表。已知曲线按名称列出, 未知曲线以十六进制显示, 例如:

```
0x001d:prime256v1:secp521r1:secp384r1
```

#### 重要

该变量仅在使用 OpenSSL 1.0.2 或更高版本时支持。在较旧版本中, 变量值将为空字符串。

该变量仅适用于新会话。

`$ssl_protocol`

返回已建立 SSL 连接的协议。

`$ssl_server_cert_type`

取值为 RSA, DSA, ECDSA, ED448, ED25519, SM2, RSA-PSS, 或 unknown, 取决于服务器证书和密钥的类型。

`$ssl_server_name`

返回通过 SNI 请求的服务器名称。

`$ssl_session_id`

返回已建立 SSL 连接的会话标识符。

`$ssl_session_reused`

返回 r 如果 SSL 会话被重用, 否则返回""

### SSL 预读取

启用从 ClientHello 消息中提取信息而不终止 SSL/TLS, 例如通过 SNI 请求的服务器名称或在 ALPN 中宣传的协议。

当从源代码构建时, 默认情况下不会构建此模块; 它应该通过 `--with-stream_ssl_preread_module` 构建选项启用。

在来自 我们的仓库的包和映像中, 该模块已包含在构建中。

## 配置示例

### 通过服务器名称选择上游

```
map $ssl_preread_server_name $name {
    backend.example.com    backend;
    default                backend2;
}

upstream backend {
    server 192.168.0.1:12345;
    server 192.168.0.2:12345;
}

upstream backend2 {
    server 192.168.0.3:12345;
    server 192.168.0.4:12345;
}

server {
    listen      12346;
    proxy_pass  $name;
    ssl_preread on;
}
```

### 通过协议选择服务器

```
map $ssl_preread_alpn_protocols $proxy {
    ~\bh2\b          127.0.0.1:8001;
    ~\bhttp/1.1\b    127.0.0.1:8002;
    ~\bxmpp-client\b 127.0.0.1:8003;
}

server {
    listen      9000;
    proxy_pass  $proxy;
    ssl_preread on;
}
```

## 通过 SSL 版本选择服务器

```
map $ssl_preread_protocol $upstream {  
    ""          ssh.example.com:22;  
    "TLSv1.2"  new.example.com:443;  
    default    tls.example.com:443;  
}  
  
# ssh 和 https 在同一端口  
server {  
    listen      192.168.0.1:443;  
    proxy_pass  $upstream;  
    ssl_preread on;  
}
```

## 指令

### ssl\_preread

|     |                       |
|-----|-----------------------|
| 语法  | ssl_preread on   off; |
| 默认  | ssl_preread off;      |
| 上下文 | stream, server        |

在预读取阶段启用从 ClientHello 消息中提取信息。

### 内置变量

`$ssl_preread_protocol`

客户端支持的最高 SSL 版本。

`$ssl_preread_server_name`

通过 SNI 请求的服务器名称。

`$ssl_preread_alpn_protocols`

客户端通过 ALPN 宣传的协议列表。这些值用逗号分隔。

## 上游

该模块用于定义可以通过 `proxy_pass` 指令引用的服务器组。

## 配置示例

```
upstream backend {
    hash $remote_addr consistent;
    zone backend 1m;

    server backend1.example.com:1935 weight=5;
    server unix:/tmp/backend3;
    server backend3.example.com service=_example._tcp resolve;

    server backup1.example.com:1935 backup;
    server backup2.example.com:1935 backup;
}

resolver 127.0.0.53 status_zone=resolver;

server {
    listen 1936;
    proxy_pass backend;
}
```

## 指令

### upstream

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>upstream name { ... }</code> |
| 默认值 | —                                  |
| 上下文 | stream                             |

定义一组服务器。服务器可以监听不同的端口。此外，监听 TCP 和 UNIX 域套接字的服务器可以混合使用。

示例：



```
upstream backend {  
    server backend1.example.com:1935 weight=5;  
    server 127.0.0.1:1935 max_fails=3 fail_timeout=30s;  
    server unix:/tmp/backend2;  
    server backend3.example.com:1935 resolve;  
  
    server backup1.example.com:1935 backup;  
}
```

默认情况下, 请求在服务器之间使用加权轮询负载均衡方法进行分配。在上面的示例中, 每 7 个请求将按如下方式分配: 5 个请求发送到 backend1.example.com, 剩下的请求分别发送到第二和第三个服务器。

如果在与服务器通信时发生错误, 请求将被传递到下一个服务器, 依此类推, 直到尝试所有可用的服务器为止。如果所有服务器都无法获得成功的响应, 客户端将收到与最后一个服务器通信的结果。

### server

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>server address [parameters];</code> |
| 默认值 | —                                         |
| 上下文 | upstream                                  |

定义服务器的地址和其他参数。地址可以指定为带有必需端口的域名或 IP 地址, 或在 `unix:` 前缀后指定为 UNIX 域套接字路径。解析为多个 IP 地址的域名一次定义多个服务器。

可以定义以下参数:

|                               |                                                                |
|-------------------------------|----------------------------------------------------------------|
| <code>weight=number</code>    | 设置服务器的权重默认值为 1。                                                |
| <code>max_conns=number</code> | 限制到代理服务器的最大同时活动连接数。默认值为 0, 表示没有限制。如果服务器组不在共享内存中, 则限制适用于每个工作进程。 |

`max_fails=number` — 设置在 `fail_timeout` 指定的时间段内与服务器通信失败的尝试次数, 以便认为服务器不可用; 然后将在同一时间段后重试。

此处, 不成功的尝试是指在与服务器建立连接时发生的错误或超时。

#### 备注

如果上游中的 `server` 解析为多个对等体, 其 `max_fails` 设置将分别应用于每个对等体。

如果在解析完所有 `server` 指令后, 上游中只有一个对等体, `max_fails` 设置将无效并被忽略。

|                          |           |
|--------------------------|-----------|
| <code>max_fails=1</code> | 默认不成功尝试次数 |
| <code>max_fails=0</code> | 禁用尝试的计数   |

`fail_timeout=time` —设置在指定时间段内应发生的与服务器通信失败次数 (`max_fails`) 以便认为服务器不可用。然后服务器在同一时间段内变得不可用, 之后再重试。

默认设置为 10 秒。

#### 备注

如果上游中的 `server` 解析为多个对等体, 其 `fail_timeout` 设置将分别应用于每个对等体。

如果在解析完所有 `server` 指令后, 上游中只有一个对等体, `fail_timeout` 设置将无效并被忽略。

|                          |                                                                                    |
|--------------------------|------------------------------------------------------------------------------------|
| <code>backup</code>      | 将服务器标记为备份服务器。当主服务器不可用时, 将传递请求给它。                                                   |
| <code>down</code>        | 将服务器标记为永久不可用。                                                                      |
| <code>drain (PRO)</code> | 将服务器设为排出状态; 这意味着它仅接收早先通过 <code>sticky</code> 绑定的会话请求。否则它的行为类似于 <code>down</code> 。 |

#### 小心

`backup` 参数不能与 `hash` 和 `random` 负载均衡方法一起使用。

`down` 和 `drain` 选项是互斥的。

Added in version 1.3.0.

|                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>resolve</code>      | 启用对与域名对应的 IP 地址列表的更改进行监控, 更新它而无需重新加载配置。该组应存储在共享内存区中; 还需要定义 <code>resolver</code> 。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>service=name</code> | 启用解析 DNS SRV 记录并设置服务名称。要使此参数生效, 请指定 <code>resolve</code> 服务器参数, 不带端口号提供主机名。<br>如果服务名称中没有点, 名称将根据 RFC 标准形成: 服务名称以 <code>_</code> 为前缀, 然后在点后加上 <code>_tcp</code> 。因此, 服务名称 <code>http</code> 将变成 <code>_http._tcp</code> 。<br>Angie 通过组合规范化的服务名称和主机名解析 SRV 记录, 并通过 DNS 获取该组合的服务器列表及其优先级和权重。 <ul style="list-style-type: none"><li>最高优先级的 SRV 记录 (具有最小优先级值的记录) 解析为主服务器, 其他记录变为备份服务器。如果在 <code>server</code> 中设置了 <code>backup</code>, 最高优先级的 SRV 记录解析为备份服务器, 其他记录将被忽略。</li><li>权重影响通过分配的容量选择服务器: 权重越高, 收到的请求越多。如果由 <code>server</code> 指令和 SRV 记录同时设置权重, 则使用 <code>server</code> 设置的权重。</li></ul> |

此示例将查找 `_http._tcp.backend.example.com` 记录:

```
server backend.example.com service=http resolve;
```

Added in version 1.4.0.

`slow_start=time` 设置服务器重新上线时恢复 `weight` 的时间, 如果负载均衡使用 `round-robin` 或 `least_conn` 方法。  
如果设置了该值, 并且服务器再次被视为可用和健康 (由 `max_fails` 和 `upstream_probe (PRO)` 定义), 则服务器将在分配的时间内稳步恢复其指定的权重。  
如果未设置该值, 则处于类似情况的服务器将立即恢复其指定的权重。

#### 备注

如果在上游中只有一个 `server`, 则 `slow_start` 无效并将被忽略。

## state (PRO)

Added in version 1.4.0: PRO

|     |                          |
|-----|--------------------------|
| 语法  | <code>state file;</code> |
| 默认值 | —                        |
| 上下文 | upstream                 |

指定上游的服务器列表持久化的 `file`。从我们的包安装时, 会创建一个具有适当权限的指定 `/var/lib/angie/state/` (`/var/db/angie/state/` 在 FreeBSD 上) 目录来存储这些文件, 因此您只需要在配置中添加文件的基本名称:

```
upstream backend {  
  
    zone backend 1m;  
    state /var/lib/angie/state/<FILE NAME>;  
}
```

服务器列表的格式类似于 `server`。每当通过配置 API 对 `/config/stream/upstreams/` 部分中的服务器进行任何修改时, 文件内容就会发生变化。文件在 Angie 启动或配置重新加载时读取。

#### 小心

要在 `upstream` 块中使用 `state` 指令, 该块不应有 `server` 指令; 相反, 它必须有一个共享内存区 (`zone`)。

## zone

|     |                                |
|-----|--------------------------------|
| 语法  | <code>zone name [size];</code> |
| 默认值 | —                              |
| 上下文 | upstream                       |

定义共享内存区的名称和大小, 该内存区保留组的配置和在工作进程之间共享的运行状态。多个组可以共享同一区域。在这种情况下, 只需指定一次大小即可。

## feedback (PRO)

Added in version 1.7.0: PRO

|     |                                                                                                    |
|-----|----------------------------------------------------------------------------------------------------|
| 语法  | <code>feedback variable [inverse] [factor=number] [account=condition_variable] [last_byte];</code> |
| 默认值 | —                                                                                                  |
| 上下文 | upstream                                                                                           |

为 `upstream` 启用基于反馈的负载均衡机制。它动态调整负载均衡决策, 将每个对等体的权重乘以其平均反馈值, 该值受 `variable` 的时间值影响, 并受一个可选条件的约束。

接受以下参数:

|                       |                                                                                                                                                    |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>variable</code> | 从中获取反馈值的变量。它应代表性能或健康指标, 并由对等体提供。该值在每次从对等体响应时进行评估, 并根据 <code>inverse</code> 和 <code>factor</code> 设置纳入滚动平均值。                                        |
| <code>inverse</code>  | 如果设置, 则反馈值被反向解释, 意味着较低的值表示更好的性能。                                                                                                                   |
| <code>factor</code>   | 反馈值在计算平均值时的加权因子。有效值是介于 0 和 99 之间的整数。默认值为 -90。<br>平均反馈使用 <a href="#">指数移动平均</a> 公式计算。<br>因子越大, 平均值受新值的影响越小; 如果因子设置为 90, 结果包含 90% 的前值 and 仅 10% 的新值。 |
| <code>account</code>  | 指定一个条件变量, 该变量控制如何将连接纳入计算。只有当条件变量不为 "" 或 "0" 时, 平均值才会更新反馈值。                                                                                         |

### 备注

默认情况下, 来自探测的流量不包括在计算中; 结合 `$upstream_probe` 变量与 `account` 可以将其包括在内, 或甚至排除其他所有内容。

示例:

```
upstream backend {  
  
    zone backend 1m;  
  
    feedback $feedback_value factor=80 account=$condition_value;  
  
    server backend1.example.com:1935 weight=1;  
    server backend2.example.com:1935 weight=2;  
}  
  
map $protocol $feedback_value {  
    "TCP"                100;  
    "UDP"                75;  
    default              10;  
}  
  
map $upstream_probe $condition_value {  
    "high_priority" "1";  
    "low_priority"  "0";  
    default         "1";  
}
```

这将根据不同会话使用的具体协议将服务器分类到不同的反馈级别，并且还添加了一个从`$upstream_probe`映射的条件，仅考虑 `high_priority` 探测或常规客户端会话。

## hash

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>hash key [consistent];</code> |
| 默认  | —                                   |
| 上下文 | upstream                            |

指定服务器组的负载均衡方法，其中客户端-服务器映射基于哈希键值。键可以包含文本、变量及其组合 (1.11.2)。使用示例：

```
hash $remote_addr;
```

注意，添加或移除服务器可能导致重新映射大多数键到不同的服务器。该方法兼容 `Cache::Memcached` Perl 库。

如果指定了 `consistent` 参数，将使用 `ketama` 一致性哈希方法。该方法确保仅有少数键在服务器添加或移除时重新映射到不同的服务器，这有助于提高缓存服务器的缓存命中率。该方法兼容 `Cache::Memcached::Fast` Perl 库，且 `ketama_points` 参数设置为 160。

## least\_conn

|     |             |
|-----|-------------|
| 语法  | least_conn; |
| 默认  | —           |
| 上下文 | upstream    |

指定一个组使用负载均衡方法, 其中连接传递到活动连接数最少的服务器, 同时考虑服务器的权重。如果有多个这样的服务器, 则通过加权轮询负载均衡方法轮流尝试。

## least\_time (PRO)

|     |                                                                                              |
|-----|----------------------------------------------------------------------------------------------|
| 语法  | least_time connect   first_byte   last_byte [factor=number]<br>[account=condition_variable]; |
| 默认  | —                                                                                            |
| 上下文 | upstream                                                                                     |

设置负载均衡方法, 其中连接转发到活动服务器的概率与响应所需的平均时间成反比; 响应时间越短, 服务器接收的连接越多。

|            |                     |
|------------|---------------------|
| connect    | 指令考虑建立连接所需的平均时间。    |
| first_byte | 指令使用接收响应第一个字节的平均时间。 |
| last_byte  | 指令使用接收完整响应的平均时间。    |

Added in version 1.7.0: PRO

|         |                                                      |
|---------|------------------------------------------------------|
| factor  | 用于与 <i>response_time_factor (PRO)</i> 相同目的, 并在设置时覆盖。 |
| account | 指定控制哪些连接应包含在计算中的条件变量。平均值仅在连接的条件变量不为 "" 或 "0" 时更新。    |

### 备注

默认情况下, 探测 不包含在计算中; 结合 *\$upstream\_probe* 变量与 *account* 可以将其包括在内, 或甚至排除其他所有内容。

相应的移动平均值, 经过 *factor* 和 *account* 调整后, 也在服务器的 *health* 对象中作为 *connect\_time*、*first\_byte\_time* 和 *last\_byte\_time* 呈现于 API 的 *stream upstream metrics* 中。

## random

|     |                            |
|-----|----------------------------|
| 语法  | <code>random [two];</code> |
| 默认  | —                          |
| 上下文 | upstream                   |

指定一个组使用负载均衡方法, 其中请求传递到随机选择的服务器, 同时考虑服务器的权重。

可选的 `two` 参数指示 Angie 随机选择两个服务器, 然后使用指定的方法选择一个服务器。默认方法是 `least_conn`, 将请求传递到活动连接数最少的服务器。

## response\_time\_factor (PRO)

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>response_time_factor number;</code> |
| 默认  | <code>response_time_factor 90;</code>     |
| 上下文 | upstream                                  |

设置 `least_time (PRO)` 负载均衡方法的平滑因子, 根据 [指数加权移动平均](#) 的公式, 使用 **前一个**值来计算平均响应时间。

指定的 `number` 越大, 新值对平均值的影响越小; 如果指定了 90, 则取前值的 90%, 新值仅占 10%。可接受值范围从 0 到 99 (含)。

相应的移动平均值在服务器的 `health` 对象中作为 `connect_time` (建立连接所需时间)、`first_byte_time` (接收响应第一个字节所需时间) 和 `last_byte_time` (接收完整响应所需时间) 呈现于 API 的 `stream upstream metrics` 中。

### 备注

仅成功响应会被考虑在计算中; 什么构成不成功的响应由 `proxy_next_upstream` 指令决定。

## sticky

Added in version 1.6.0: Angie

Added in version 1.6.0: Angie PRO

|     |                                                                                                                                                                           |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>sticky route \$variable...;</code><br><code>sticky learn zone=zone create=\$create_var1... lookup=\$lookup_var1... [connect]</code><br><code>[timeout=time];</code> |
| 默认  | —                                                                                                                                                                         |
| 上下文 | upstream                                                                                                                                                                  |

配置客户端会话绑定到代理服务器的模式; 为了从定义了 `sticky` 的服务器中排空请求, 使用 `drain` 选项在 `server` 块中。

### ⚠ 注意

`sticky` 指令必须在所有设置负载均衡方法的指令之后使用; 否则将不起作用。

### route 模式

此模式使用预定义的路由标识符, 可以嵌入 Angie 可以访问的任何连接属性中。它的灵活性较低, 因为它依赖于预定义的值, 但如果这些标识符已存在, 则可能更适合。

在此, 当与代理服务器建立连接时, 它可以为客户端分配一个路由, 并以双方都知悉的方式返回其标识符。 `server` 指令的 `sid` 参数的值必须用作路由标识符。注意, 如果设置了 `sticky_secret` 指令, 该参数会被额外哈希。

希望使用该路由的客户端的后续连接必须包含服务器发出的标识符, 确保其最终位于 Angie 变量中。

指令列出了用于路由的特定变量。选择传入连接路由到的服务器时, 使用第一个非空变量; 然后将其与 `server` 指令的 `sid` 参数进行比较。如果选择服务器失败或所选服务器无法接受连接, 则根据配置的负载均衡方法选择另一个服务器。

在此, Angie 在自定义 `$route` 变量中查找标识符, 该变量从 `$ssl_preread_server_name` 映射 (注意必须启用 `ssl_preread`):

```
stream {  
  
    map $ssl_preread_server_name $route {  
  
        a.example.com      a;  
        b.example.com      b;  
        default            "";  
    }  
  
    upstream backend {  
  
        server 127.0.0.1:8081 sid=a;  
        server 127.0.0.1:8082 sid=b;  
  
        sticky route $route;  
    }  
  
    server {  
  
        listen 127.0.0.1:8080;  
  
        ssl_preread on;  
  
        proxy_pass backend;  
    }  
}
```



```
}  
}
```

### learn 模式 (PRO)

此模式使用动态生成的密钥将客户端与特定代理服务器关联; 它更灵活, 因为它在运行时分配服务器, 在共享内存区域存储会话, 并支持不同的会话标识符传递方式。

在此, 基于来自代理服务器的连接属性创建会话。create 和 lookup 参数列出表示如何创建新会话和查找现有会话的变量。两个参数可以多次出现。

会话标识符是第一个非空变量的值, 该变量由 create 指定; 例如, 这可以是代理服务器的名称。

会话存储在共享内存区域中; 其名称和大小由 zone 参数设置。如果会话在 timeout 设置的时间内保持不活动, 则会被删除。默认是 1 小时。

希望使用该会话的客户端的后续连接必须包含其标识符, 确保其最终位于 lookup 指定的非空变量中; 其值将与共享内存中的会话匹配。如果选择服务器失败或所选服务器无法接受连接, 则根据配置的负载均衡方法选择另一个服务器。

connect 参数允许在与代理服务器建立连接后立即创建会话。否则, 仅在处理连接后创建会话。

在示例中, Angie 使用 \$rdp\_cookie 变量创建和查找会话:

```
stream {  
  
    upstream backend {  
  
        server 127.0.0.1:3390 sid=a;  
        server 127.0.0.1:3391 sid=b;  
  
        sticky learn lookup=$rdp_cookie create=$rdp_cookie zone=sessions:1m;  
    }  
  
    server {  
  
        listen 127.0.0.1:3389;  
  
        ssl_preread on;  
  
        proxy_pass backend;  
    }  
}
```

### sticky\_strict

Added in version 1.6.0: Angie

Added in version 1.6.0: Angie PRO

|         |                                      |
|---------|--------------------------------------|
| Syntax  | <code>sticky_strict on   off;</code> |
| Default | <code>sticky_strict off;</code>      |
| Context | upstream                             |

启用时, 如果所需的服务器不可用, Angie 会向客户端返回连接错误, 而不是使用其他可用的服务器, 正如当上游中没有服务器可用时那样。

### sticky\_secret

Added in version 1.6.0: Angie

Added in version 1.6.0: Angie PRO

|         |                                    |
|---------|------------------------------------|
| Syntax  | <code>sticky_secret string;</code> |
| Default | —                                  |
| Context | upstream                           |

将 *string* 添加为 MD5 哈希函数的盐值, 用于 `route` 模式下的 *sticky* 指令。 *string* 可以包含变量, 例如 `$remote_addr`:

```
upstream backend {
    server 127.0.0.1:8081 sid=a;
    server 127.0.0.1:8082 sid=b;

    sticky route $route;
    sticky_secret my_secret.$remote_addr;
}
```

盐值附加在被哈希的值上; 要独立验证哈希机制:

```
$ echo -n "<VALUE><SALT>" | md5sum
```

## 内置变量

`stream_upstream` 模块支持以下内置变量:

### `$upstream_addr`

保持上游服务器的 IP 地址和端口, 或 UNIX 域套接字的路径。如果在请求处理期间联系了多个服务器, 其地址以逗号分隔, 例如:

```
192.168.1.1:1935, 192.168.1.2:1935, unix:/tmp/sock
```

如果无法选择服务器, 变量保存 *server group* 的 *name*。

### `$upstream_bytes_received`

从上游服务器接收的字节数。多个连接的值以逗号分隔, 类似于 `$upstream_addr` 变量中的地址。

### `$upstream_bytes_sent`

发送到上游服务器的字节数。多个连接的值以逗号分隔, 类似于 `$upstream_addr` 变量中的地址。

### `$upstream_connect_time`

连接上游服务器的时间; 时间以秒为单位, 精确到毫秒。多个连接的时间以逗号分隔, 类似于 `$upstream_addr` 变量中的地址。

### `$upstream_first_byte_time`

接收第一个字节数据的时间; 时间以秒为单位, 精确到毫秒。多个连接的时间以逗号分隔, 类似于 `$upstream_addr` 变量中的地址。

### `$upstream_session_time_<name>`

会话持续时间, 以秒为单位, 精确到毫秒。多个连接的时间以逗号分隔, 类似于 `$upstream_addr` 变量中的地址。

`$upstream_sticky_status`

粘性连接的状态。

|      |                          |
|------|--------------------------|
| ""   | 连接路由到上游而未启用粘性。           |
| NEW  | 无粘性信息的连接。                |
| HIT  | 含粘性信息的连接路由到所需的后端。        |
| MISS | 含粘性信息的连接路由到由负载均衡算法选择的后端。 |

多个连接的值以逗号和冒号分隔, 类似于 `$upstream_addr` 变量中的地址。

### 上游探针

该模块为 `stream_upstream` 实现主动健康探针。

### 配置示例

```
server {
    listen ...;

    # ...
    proxy_pass backend;
    upstream_probe_timeout 1s;

    upstream_probe backend_probe
        port=12345
        interval=5s
        test=$good
        essential
        fails=3
        passes=3
        max_response=512k
        mode=onfail
        "send=data:GET / HTTP/1.0\n\n";
}
```

## 指令

### upstream\_probe (PRO)

Added in version 1.4.0: PRO

|     |                                                                                                                                                                                                                |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>upstream_probe name [port=number] [interval=time] [test=condition] [essential [persistent]] [fails=number] [passes=number] [max_response=size] [mode=always   idle   onfail] [udp] [send=string];</code> |
| 默认  | —                                                                                                                                                                                                              |
| 上下文 | server                                                                                                                                                                                                         |

定义一个针对 *upstream* 组内节点的主动健康探针, 这些组在相同 *location* 上下文中通过 *upstream\_probe* 指令为 *proxy\_pass* 指定。随后, Angie 会根据此处配置的参数定期探测上游组的每个节点。

如果向节点的请求成功, 并考虑到 *upstream\_probe* 指令的所有参数设置以及控制指令 *location* 上下文使用上游的设置, 包括 *proxy\_next\_upstream* 指令, 则节点的探测通过。

为了使用探针, 上游必须具有共享内存区域 (*zone*)。一个上游可以配置多个探针。

接受以下参数:

|                     |                                                                                                                                                                                                                                |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>name</b>         | 探针的必需名称。                                                                                                                                                                                                                       |
| <b>port</b>         | 探针请求的备用端口号。                                                                                                                                                                                                                    |
| <b>interval</b>     | 探针之间的间隔。默认值—5s。                                                                                                                                                                                                                |
| <b>test</b>         | 探针的条件, 定义为变量字符串。如果变量替换结果为 "" 或 "0", 则探针不通过。                                                                                                                                                                                    |
| <b>essential</b>    | 如果设置, 则在探针通过之前, 节点不会接收客户端请求。                                                                                                                                                                                                   |
| <b>persistent</b>   | 设置此参数需要先启用 <i>essential</i> ; 被认为健康的 <i>persistent</i> 节点在配置重载之前开始接收请求, 而无需先通过此探针。                                                                                                                                             |
| <b>fails</b>        | 使节点不健康的连续失败探针次数。默认值—1。                                                                                                                                                                                                         |
| <b>passes</b>       | 使节点健康的连续通过探针次数。默认值—1。                                                                                                                                                                                                          |
| <b>max_response</b> | 响应的最大内存大小。如果指定零值, 则禁用响应等待。默认值—256k。                                                                                                                                                                                            |
| <b>mode</b>         | 探针模式, 取决于节点的健康状态: <ul style="list-style-type: none"> <li>• <i>always</i> —无论状态如何都探测节点;</li> <li>• <i>idle</i> —探测不健康的节点和自上次客户端请求以来经过 <i>interval</i> 的节点。</li> <li>• <i>onfail</i> —仅探测不健康的节点。</li> </ul> 默认值— <i>always</i> 。 |
| <b>udp</b>          | 如果指定, 则使用 UDP 协议进行探测。默认情况下, 使用 TCP 进行探测。                                                                                                                                                                                       |
| <b>send</b>         | 用于检查的数据; 这可以是带有前缀 <i>data:</i> 的字符串或带有数据的文件名 (绝对指定或相对于 <i>/usr/local/angie/</i> 目录)。                                                                                                                                           |

示例:

```
upstream backend {
    zone backend 1m;

    server a.example.com;
    server b.example.com;
}

map $upstream_probe_response $good {
    ~200    "1";
    default  "";
}

server {
    listen ...;

    # ...

    proxy_pass backend;
    upstream_probe_timeout 1s;

    upstream_probe backend_probe
        port=12345
        interval=5s
        test=$good
        essential
        persistent
        fails=3
        passes=3
        max_response=512k
        mode=onfail
        "send=data:GET / HTTP/1.0\n\n";
}
```

探针操作细节:

- 最初, 节点在通过为其配置的所有 `essential` 探针之前不会接收客户端请求, 如果配置被重载并且节点在此之前被认为是健康的, 则跳过 `persistent` 探针。如果没有这样的探针, 则认为节点是健康的。
- 如果为节点配置的任何探针达到 `fails` 或节点达到 `max_fails`, 则该节点被认为是不健康的, 不会接收客户端请求。
- 为了使不健康的节点再次被认为是健康的, 为其配置的所有探针必须达到各自的 `passes`; 之后, 还会考虑 `max_fails`。

## 内置变量

`stream_upstream` 模块支持以下内置变量:

### `$upstream_probe` (PRO)

当前活动的 `upstream_probe` 的名称。

### `$upstream_probe_response` (PRO)

在由 `upstream_probe` 配置的主动探针期间接收到的响应内容。

核心流模块实现了处理 TCP 和 UDP 连接的基础功能, 包括定义服务器块、流量路由、配置代理、SSL/TLS 支持及管理用于数据库、DNS 和其他基于 TCP 和 UDP 协议的流服务的连接。

本节的其他模块扩展了这些功能, 使您可以灵活配置和优化流服务器以适应各种场景和要求。

当从源代码构建时, 该模块默认不构建; 应使用 `--with-stream` 构建选项启用。在我们的仓库提供的包和镜像中, 该模块已包含在构建中。

## 配置示例

```
worker_processes auto;

error_log /var/log/nginx/error.log info;

events {
    worker_connections 1024;
}

stream {
    upstream backend {
        hash $remote_addr consistent;

        server backend1.example.com:12345 weight=5;
        server 127.0.0.1:12345 max_fails=3 fail_timeout=30s;
        server unix:/tmp/backend3;
    }

    upstream dns {
        server 192.168.0.1:53535;
        server dns.example.com:53;
    }

    server {
        listen 12345;
```

```
    proxy_connect_timeout 1s;
    proxy_timeout 3s;
    proxy_pass backend;
}

server {
    listen 127.0.0.1:53 udp reuseport;
    proxy_timeout 20s;
    proxy_pass dns;
}

server {
    listen [::1]:12345;
    proxy_pass unix:/tmp/stream.socket;
}
}
```

## 指令

### listen

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>listen</code> <i>address[:port]</i> [ <i>ssl</i> ] [ <i>udp</i> ] [ <i>proxy_protocol</i> ] [ <i>setfib=number</i> ]<br>[ <i>fastopen=number</i> ] [ <i>backlog=number</i> ] [ <i>rcvbuf=size</i> ] [ <i>sndbuf=size</i> ]<br>[ <i>accept_filter=filter</i> ] [ <i>deferred</i> ] [ <i>bind</i> ] [ <i>ipv6only=on   off</i> ] [ <i>reuseport</i> ]<br>[ <i>so_keepalive=on off</i> ][ <i>keepidle</i> ]:[ <i>keepintvl</i> ]:[ <i>keepcnt</i> ]; |
| 默认值 | —                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 上下文 | server                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

设置服务器接受连接的 *address* 和 *port*。可以仅指定 *port*。地址也可以是主机名，例如：

```
listen 127.0.0.1:12345;
listen *:12345;
listen 12345;      # 与 *:12345 相同
listen localhost:12345;
```

IPv6 地址需要用方括号括起来：

```
listen [::1]:12345;
listen [::]:12345;
```

UNIX 域套接字需要用 `unix:` 前缀指定：

```
listen unix:/var/run/angie.sock;
```

端口范围由第一个和最后一个端口用连字符分隔指定：



```
listen 127.0.0.1:12345-12399;  
listen 12345-12399;
```

### 重要

不同的服务器必须监听不同的 *address:port* 对。

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| ssl            | 指定该端口上接受的所有连接都应以 SSL 模式工作。                                       |
| udp            | 为处理数据报配置监听套接字。为了在同一会话中处理来自相同地址和端口的数据包, 还应指定 <i>reuseport</i> 参数。 |
| proxy_protocol | 指定该端口上接受的所有连接应使用 PROXY 协议。                                       |

*listen* 指令可以具有特定于与套接字相关的系统调用的多个附加参数。

|                        |                                                      |
|------------------------|------------------------------------------------------|
| <i>setfib=number</i>   | 为监听套接字设置关联的路由表, FIB (SO_SETFIB 选项)。目前仅在 FreeBSD 上工作。 |
| <i>fastopen=number</i> | 为监听套接字启用“TCP Fast Open”并限制尚未完成三次握手的连接队列的最大长度。        |

### 小心

不要启用此功能, 除非服务器可以处理多次接收带有数据的相同 SYN 包的情况。

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>backlog=number</code>       | 在 <code>listen()</code> 调用中设置 <code>backlog</code> 参数, 它限制了等待连接队列的最大长度。默认情况下, <code>backlog</code> 在 FreeBSD、DragonFly BSD 和 macOS 上设置为 <code>-1</code> , 在其他平台上设置为 <code>511</code> 。                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>rcvbuf=size</code>          | 为监听套接字设置接收缓冲区大小 ( <code>SO_RCVBUF</code> 选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <code>sndbuf=size</code>          | 为监听套接字设置发送缓冲区大小 ( <code>SO_SNDBUF</code> 选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <code>accept_filter=filter</code> | 为监听套接字设置接受过滤器的名称 ( <code>SO_ACCEPTFILTER</code> 选项), 在将传入连接传递给 <code>accept()</code> 之前过滤它们。这仅在 FreeBSD 和 NetBSD 5.0+ 上有效。可接受的值为 <code>dataready</code> 和 <code>httpready</code> 。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <code>deferred</code>             | 指示在 Linux 上使用延迟的 <code>accept()</code> ( <code>TCP_DEFER_ACCEPT</code> 套接字选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <code>bind</code>                 | 该参数指示为给定的 <code>address:port</code> 对进行单独的 <code>bind()</code> 调用。事实是, 如果有多个 <code>listen</code> 指令具有相同的 <code>port</code> 但不同的地址, 并且其中一个 <code>listen</code> 指令监听给定端口的所有地址 ( <code>*:port</code> ), <code>Angie</code> 将仅对 <code>*:port</code> 进行 <code>bind()</code> 。需要注意的是, 在这种情况下, 将进行 <code>getsockname()</code> 系统调用以确定接受连接的地址。如果使用了 <code>setfib</code> 、 <code>fastopen</code> 、 <code>backlog</code> 、 <code>rcvbuf</code> 、 <code>sndbuf</code> 、 <code>accept_filter</code> 、 <code>deferred</code> 、 <code>ipv6only</code> 、 <code>reuseport</code> 或 <code>so_keepalive</code> 参数, 则将始终为给定的 <code>address:port</code> 对进行单独的 <code>bind()</code> 调用。 |
| <code>ipv6only=on   off</code>    | 该参数确定 (通过 <code>IPV6_V6ONLY</code> 套接字选项) 监听通配地址 <code>:::</code> 的 IPv6 套接字是否仅接受 IPv6 连接或同时接受 IPv6 和 IPv4 连接。该参数默认开启。启动时只能设置一次。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>reuseport</code>            | 该参数指示为每个工作进程创建一个单独的监听套接字 (在 Linux 3.9+ 和 DragonFly BSD 上使用 <code>SO_REUSEPORT</code> 套接字选项, 或在 FreeBSD 12+ 上使用 <code>SO_REUSEPORT_LB</code> ), 允许内核在工作进程之间分配传入连接。目前仅在 Linux 3.9+、DragonFly BSD 和 FreeBSD 12+ 上工作。                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

### 小心

不当使用此选项可能会产生安全隐患。

`so_keepalive=on | off | [keepidle]:[keepintvl]:[keepcnt]` 配置监听套接字的“TCP keepalive”行为。

|                  |                                     |
|------------------|-------------------------------------|
| <code>''</code>  | 如果省略此参数, 则套接字将使用操作系统的默认设置           |
| <code>on</code>  | 为套接字开启 <code>SO_KEEPALIVE</code> 选项 |
| <code>off</code> | 为套接字关闭 <code>SO_KEEPALIVE</code> 选项 |

一些操作系统支持使用 `TCP_KEEPIDLE`、`TCP_KEEPINTVL` 和 `TCP_KEEPCNT` 套接字选项在每个套接字基础上设置 TCP keepalive 参数。在这样的系统 (目前是 Linux 2.4+、NetBSD 5+ 和 FreeBSD 9.0-STABLE) 上, 可以使用 `keepidle`、`keepintvl` 和 `keepcnt` 参数进行配置。可以省略一个或两个参数, 在这种情况下, 将使用相应套接字选项的系统默认设置。

例如,

```
so_keepalive=30m::10
```

将空闲超时 (`TCP_KEEPIDLE`) 设置为 30 分钟, 保留探测间隔 (`TCP_KEEPINTVL`) 为系统默认, 并将探测次数 (`TCP_KEEPCNT`) 设置为 10 次。

### preread\_buffer\_size

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>preread_buffer_size size;</code> |
| 默认值 | <code>preread_buffer_size 16k;</code>  |
| 上下文 | stream, server                         |

指定 `preread` 缓冲区的大小。

### preread\_timeout

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>preread_timeout timeout;</code> |
| 默认值 | <code>preread_timeout 30s;</code>     |
| 上下文 | stream, server                        |

指定 `preread` 阶段的超时时间。

### proxy\_protocol\_timeout

|     |                                              |
|-----|----------------------------------------------|
| 语法  | <code>proxy_protocol_timeout timeout;</code> |
| 默认值 | <code>proxy_protocol_timeout 30s;</code>     |
| 上下文 | stream, server                               |

指定读取 PROXY 协议头的 `timeout` 时间。如果在此时间内未传输完整的头, 则连接将被关闭。

### resolver

|     |                                                                                                        |
|-----|--------------------------------------------------------------------------------------------------------|
| 语法  | <code>resolver address ... [valid=time] [ipv4=on   off] [ipv6=on   off]<br/>[status_zone=zone];</code> |
| 默认值 | —                                                                                                      |
| 上下文 | stream, server, upstream                                                                               |

配置用于将上游服务器的名称解析为地址的名称服务器, 例如:

```
resolver 127.0.0.53 [::1]:5353;
```

地址可以指定为域名或 IP 地址, 并且可以选择指定端口。如果未指定端口, 则使用端口 53。名称服务器按轮询方式查询。

默认情况下, Angie 使用响应的 TTL 值缓存答案。可选的 `valid` 参数允许重写它:

|                    |                                       |
|--------------------|---------------------------------------|
| <code>valid</code> | 可选的 <code>valid</code> 参数允许重写缓存条目的有效性 |
|--------------------|---------------------------------------|

```
resolver 127.0.0.53 [::1]:5353 valid=30s;
```

默认情况下, Angie 在解析时将查找 IPv4 和 IPv6 地址。

|                       |               |
|-----------------------|---------------|
| <code>ipv4=off</code> | 禁用 IPv4 地址的查找 |
| <code>ipv6=off</code> | 禁用 IPv6 地址的查找 |

|                          |                                                                                    |
|--------------------------|------------------------------------------------------------------------------------|
| <code>status_zone</code> | 可选参数; 在指定区域内启用 DNS 服务器请求和响应指标 ( <code>/status/resolvers/&lt;zone&gt;</code> ) 的收集。 |
|--------------------------|------------------------------------------------------------------------------------|

### 小技巧

为防止 DNS 欺骗, 建议在适当安全的可信本地网络中配置 DNS 服务器。

## resolver\_timeout

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>resolver_timeout timeout;</code> |
| 默认值 | <code>resolver_timeout 30s;</code>     |
| 上下文 | stream, server, upstream               |

设置名称解析的超时时间, 例如:

```
resolver_timeout 5s;
```

## server

|     |                             |
|-----|-----------------------------|
| 语法  | <code>server { ... }</code> |
| 默认值 | —                           |
| 上下文 | stream                      |

为服务器设置配置。

## server\_name

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>server_name name ...;</code> |
| 默认值 | <code>server_name "";</code>       |
| 上下文 | server                             |

设置虚拟服务器的名称, 例如:

```
server {
    server_name example.com www.example.com;
}
```

第一个名称成为主要服务器名称。

服务器名称可以包含星号 (\*) 以替换名称的第一个或最后一个部分:

```
server {
    server_name example.com *.example.com www.example.*;
}
```

这些名称称为通配符名称。

上述前两个示例可以合并为一个:

```
server {
    server_name .example.com;
}
```

您还可以在服务器名称中使用正则表达式, 方法是在名称前加上波浪号 (~):

```
server {
    server_name www.example.com ~^www\d+\.example\.com$;
}
```

正则表达式可以包括可用于其他指令的捕获:

```
server {
    server_name ~^(www\.)?(.+)$;

    proxy_pass www.$2:12345;
}
```

正则表达式中的命名捕获会创建可在其他指令中使用的变量:

```
server {
    server_name ~^(www\.)?(<domain>.+)$;

    proxy_pass www.$domain:12345;
}
```

如果指令的参数设置为 `$hostname`, 则插入计算机的主机名。

在按名称搜索虚拟服务器时, 如果名称与指定的多个变体 (例如, 通配符名称和正则表达式匹配) 匹配, 则将按以下优先级顺序选择第一个匹配的变体:

- 精确名称
- 以星号开头的最长通配符名称, 例如, `*.example.com`
- 以星号结尾的最长通配符名称, 例如 `mail.*`
- 第一个匹配的正则表达式 (按在配置文件中出现的顺序)

#### 注意

对于 TLS 连接, 请使用 `SSL Preread` 模块。

### server\_names\_hash\_bucket\_size

|     |                                                       |
|-----|-------------------------------------------------------|
| 语法  | <code>server_names_hash_bucket_size size;</code>      |
| 默认值 | <code>server_names_hash_bucket_size 32 64 128;</code> |
| 上下文 | stream                                                |

设置服务器名称哈希表的桶大小。默认值取决于处理器缓存行的大小。

### server\_names\_hash\_max\_size

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>server_names_hash_max_size size;</code> |
| 默认值 | <code>server_names_hash_max_size 512;</code>  |
| 上下文 | <code>stream</code>                           |

设置服务器名称哈希表的最大大小。

### status\_zone

|     |                                |
|-----|--------------------------------|
| 语法  | <code>status_zone zone;</code> |
| 默认值 | —                              |
| 上下文 | <code>server</code>            |

分配共享内存区域以收集 `/status/stream/server_zones/<zone>` 的指标。

多个 `server` 上下文可以共享同一区域进行数据收集。

### stream

|     |                             |
|-----|-----------------------------|
| 语法  | <code>stream { ... }</code> |
| 默认值 | —                           |
| 上下文 | <code>main</code>           |

提供配置文件上下文, 在该上下文中指定流服务器指令。

### tcp\_nodelay

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>tcp_nodelay on   off;</code> |
| 默认值 | <code>tcp_nodelay on;</code>       |
| 上下文 | <code>stream, server</code>        |

启用或禁用 `TCP_NODELAY` 选项。该选项适用于客户端和代理服务器连接。

### variables\_hash\_bucket\_size

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>variables_hash_bucket_size size;</code> |
| 默认值 | <code>variables_hash_bucket_size 64;</code>   |
| 上下文 | stream                                        |

设置变量哈希表的桶大小。设置哈希表的详细信息在单独的 文档中提供。

### variables\_hash\_max\_size

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>variables_hash_max_size size;</code> |
| 默认值 | <code>variables_hash_max_size 1024;</code> |
| 上下文 | stream                                     |

设置变量哈希表的最大大小。设置哈希表的详细信息在单独的 文档中提供。

## 内置变量

*stream core* 模块支持以下变量:

### `$angie_version`

Angie 版本

### `$binary_remote_addr`

客户端地址的二进制形式, 值的长度对于 IPv4 地址始终为 4 字节, 对于 IPv6 地址始终为 16 字节。

### `$bytes_received`

从客户端接收的字节数。



`$bytes_sent`

发送给客户端的字节数。

`$connection`

连接序列号。

`$hostname`

主机名。

`$msec`

当前时间, 以秒为单位, 精确到毫秒。

`$pid`

工作进程的 PID。

`$protocol`

用于与客户端通信的协议: TCP 或 UDP。

`$proxy_protocol_addr`

来自 PROXY 协议头的客户端地址必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

`$proxy_protocol_port`

来自 PROXY 协议头的客户端端口必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

### `$proxy_protocol_server_addr`

来自 PROXY 协议头的服务器地址必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

### `$proxy_protocol_server_port`

来自 PROXY 协议头的服务器端口必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

### `$proxy_protocol_tlv_<name>`

来自 PROXY 协议头的 TLV。 `name` 可以是 TLV 类型或其数值。在后者情况下, 值为十六进制且应以 `0x` 为前缀:

```
$proxy_protocol_tlv_alpn $proxy_protocol_tlv_0x01
```

SSL TLVs 也可以通过 TLV 类型名称或其数值访问, 两者都以前缀 `ssl_`:

```
$proxy_protocol_tlv_ssl_version $proxy_protocol_tlv_ssl_0x21
```

支持以下 TLV 类型名称:

- `alpn (0x01)` - 在连接上使用的上层协议。
- `authority (0x02)` - 客户端传递的主机名值。
- `unique_id (0x05)` - 唯一连接 ID。
- `netns (0x30)` - 命名空间的名称。
- `ssl (0x20)` - 二进制 SSL TLV 结构。

支持以下 SSL TLV 类型名称:

- `ssl_version (0x21)` - 客户端连接中使用的 SSL 版本。
- `ssl_cn (0x22)` - SSL 证书的公共名。
- `ssl_cipher (0x23)` - 使用的加密算法名称。
- `ssl_sig_alg (0x24)` - 用于签署证书的算法。
- `ssl_key_alg (0x25)` - 公钥算法。

此外, 还支持以下特殊 SSL TLV 类型名称:

- `ssl_verify` - 客户端 SSL 证书验证结果, 如果客户端出示了证书并成功验证则为 `0`, 否则为非零。

必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来预先启用 PROXY 协议。

`$remote_addr`

客户端地址。

`$remote_port`

客户端端口。

`$server_addr`

接受连接的服务器的地址计算此变量的值通常需要一个系统调用。为了避免系统调用, `listen` 指令必须指定地址并使用 `bind` 参数。

`$server_port`

接受连接的服务器的端口。

`$session_time`

会话持续时间, 以秒为单位, 精确到毫秒。

`$status`

会话状态, 可以是以下之一:

|     |                              |
|-----|------------------------------|
| 200 | 会话成功完成。                      |
| 400 | 无法解析客户端数据, 例如, PROXY 协议头。    |
| 403 | 禁止访问, 例如, 当对特定客户端地址 的访问有限制时。 |
| 500 | 内部服务器错误。                     |
| 502 | 错误的网关, 例如, 如果无法选择或到达上游服务器。   |
| 503 | 服务不可用, 例如, 当通过连接数量 限制访问时。    |

`$time_iso8601`

ISO 8601 标准格式的本地时间。

`$time_local`

常用日志格式的本地时间。

### 3.3.4 邮件模块

#### 认证 HTTP

##### 指令

##### `auth_http`

|     |                             |
|-----|-----------------------------|
| 语法  | <code>auth_http uri;</code> |
| 默认值 | —                           |
| 上下文 | mail, server                |

设置 HTTP 认证服务器的 URL。协议描述见下文。

##### `auth_http_header`

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>auth_http_header header value;</code> |
| 默认值 | —                                           |
| 上下文 | mail, server                                |

将指定的头部附加到发送给认证服务器的请求中。此头部可以用作共享密钥, 以验证请求是否来自 Angie。例如:

```
auth_http_header X-Auth-Key "secret_string";
```

##### `auth_http_pass_client_cert`

|     |                                                   |
|-----|---------------------------------------------------|
| 语法  | <code>auth_http_pass_client_cert on   off;</code> |
| 默认值 | <code>auth_http_pass_client_cert off;</code>      |
| 上下文 | mail, server                                      |

将"Auth-SSL-Cert" 头部与 PEM 格式 (urlencoded) 的客户端证书附加到发送给认证服务器的请求中。

## auth\_http\_timeout

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>auth_http_timeout <i>time</i>;</code> |
| 默认值 | <code>auth_http_timeout 60s;</code>         |
| 上下文 | mail, server                                |

设置与认证服务器通信的超时时间。

## 协议

HTTP 协议用于与认证服务器通信。响应体中的数据被忽略, 仅在头部传递信息。

## 请求和响应示例:

请求:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain # plain/apop/cram-md5/external
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap # imap/pop3/smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

成功响应:

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
Auth-Port: 143
```

失败响应:

```
HTTP/1.0 200 OK
Auth-Status: Invalid login or password
Auth-Wait: 3
```

如果没有“Auth-Wait”头部, 将返回错误并关闭连接。当前实现为每次认证尝试分配内存。内存仅在会话结束时释放。因此, 单个会话中的无效认证尝试次数必须有限制——在 10-20 次尝试后, 服务器必须响应不带“Auth-Wait”头部 (尝试次数在“Auth-Login-Attempt”头部中传递)。

当使用 APOP 或 CRAM-MD5 时, 请求响应如下所示:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: apop
Auth-User: user
Auth-Salt: <238188073.1163692009@mail.example.com>
Auth-Pass: auth_response
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
```

成功响应:

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: 198.51.100.1
Auth-Port: 143
Auth-Pass: plain-text-pass
```

如果响应中存在”Auth-User” 头部, 它将覆盖用于与后端认证的用户名。

对于 SMTP, 响应还会考虑”Auth-Error-Code” 头部——如果存在, 它将在错误情况下用作响应代码。否则, 535 5.7.0 代码将被添加到”Auth-Status” 头部。

例如, 如果从认证服务器收到以下响应:

```
HTTP/1.0 200 OK
Auth-Status: Temporary server problem, try again later
Auth-Error-Code: 451 4.3.0
Auth-Wait: 3
```

则 SMTP 客户端将收到错误

```
451 4.3.0 Temporary server problem, try again later
```

如果代理 SMTP 不需要认证, 请求将如下所示:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: none
Auth-User:
Auth-Pass:
Auth-Protocol: smtp
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Client-Host: client.example.org
Auth-SMTP-Helo: client.example.org
Auth-SMTP-From: MAIL FROM: <>
Auth-SMTP-To: RCPT TO: <postmaster@mail.example.com>
```

对于 SSL/TLS 客户端连接, 添加"Auth-SSL" 头部, 并且"Auth-SSL-Verify" 将包含客户端证书验证的结果, 如果启用: SUCCESS, "FAILED:reason", 以及如果没有证书则为 NONE。

当存在客户端证书时, 其详细信息将通过以下请求头部传递: "Auth-SSL-Subject", "Auth-SSL-Issuer", "Auth-SSL-Serial", 和"Auth-SSL-Fingerprint"。如果启用了`auth_http_pass_client_cert`, 证书本身将通过"Auth-SSL-Cert" 头部传递。已建立连接的协议和密码将通过"Auth-SSL-Protocol" 和"Auth-SSL-Cipher" 头部传递。请求将如下所示:

```
GET /auth HTTP/1.0
Host: localhost
Auth-Method: plain
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP: 192.0.2.42
Auth-SSL: on
Auth-SSL-Protocol: TLSv1.3
Auth-SSL-Cipher: TLS_AES_256_GCM_SHA384
Auth-SSL-Verify: SUCCESS
Auth-SSL-Subject: /CN=example.com
Auth-SSL-Issuer: /CN=example.com
Auth-SSL-Serial: C07AD56B846B5BFF
Auth-SSL-Fingerprint: 29d6a80a123d13355ed16b4b04605e29cb55a5ad
```

当使用`PROXY` 协议时, 其详细信息将通过以下请求头部传递: "Proxy-Protocol-Addr", "Proxy-Protocol-Port", "Proxy-Protocol-Server-Addr", 和"Proxy-Protocol-Server-Port"。

## IMAP

### 指令

#### imap\_auth

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>imap_auth method ...;</code> |
| 默认值 | <code>imap_auth plain;</code>      |
| 上下文 | mail, server                       |

设置 IMAP 客户端允许的认证方法。支持的方法包括:

|          |                                       |
|----------|---------------------------------------|
| plain    | LOGIN, AUTH=PLAIN                     |
| login    | AUTH=LOGIN                            |
| cram-md5 | AUTH=CRAM-MD5。为了使此方法生效, 密码必须以未加密形式存储。 |
| external | AUTH=EXTERNAL                         |

纯文本认证方法 (LOGIN 命令, AUTH=PLAIN 和 AUTH=LOGIN) 始终启用, 但如果未指定 plain 和 login 方法, AUTH=PLAIN 和 AUTH=LOGIN 将不会自动包含在 *imap\_capabilities* 中。

### imap\_capabilities

|     |                                                         |
|-----|---------------------------------------------------------|
| 语法  | <code>imap_capabilities extension ...;</code>           |
| 默认值 | <code>imap_capabilities IMAP4 IMAP4rev1 UIDPLUS;</code> |
| 上下文 | mail, server                                            |

设置在响应 CAPABILITY 命令时传递给客户端的 IMAP 协议 扩展列表。根据 *starttls* 指令的值, *imap\_auth* 指令中指定的认证方法和 STARTTLS 会自动添加到此列表中。

建议指定 IMAP 后端支持的扩展, 当客户端连接被 Angie 透明地代理到后端时 (如果这些扩展与认证后使用的命令相关)。

### imap\_client\_buffer

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>imap_client_buffer size;</code>  |
| 默认值 | <code>imap_client_buffer 4k 8k;</code> |
| 上下文 | mail, server                           |

设置用于读取 IMAP 命令的缓冲区大小。默认情况下, 缓冲区大小等于一个内存页。这取决于平台, 可能是 4K 或 8K。

## POP3

### 指令

#### pop3\_auth

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>pop3_auth method ...;</code> |
| 默认值 | <code>ipop3_auth plain;</code>     |
| 上下文 | mail, server                       |

设置 POP3 客户端允许的认证方法。支持的方法包括:

|          |                                       |
|----------|---------------------------------------|
| plain    | USER/PASS, AUTH PLAIN, AUTH LOGIN     |
| apop     | APOP。为了使此方法有效, 密码必须以未加密形式存储。          |
| cram-md5 | AUTH=CRAM-MD5。为了使此方法有效, 密码必须以未加密形式存储。 |
| external | AUTH=EXTERNAL                         |



明文认证方法 (USER/PASS、AUTH PLAIN 和 AUTH LOGIN) 始终启用, 但如果未指定 plain 方法, 则 AUTH PLAIN 和 AUTH LOGIN 将不会自动包含在 `pop3_capabilities` 中。

### pop3\_capabilities

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>pop3_capabilities extension ...;</code> |
| 默认值 | <code>pop3_capabilities TOP USER UIDL;</code> |
| 上下文 | mail, server                                  |

设置 POP3 协议扩展列表, 以响应 CAPA 命令传递给客户端。根据 `starttls` 指令的值, `pop3_auth` 指令中指定的认证方法 (SASL 扩展) 和 STLS 将自动添加到此列表中。

建议指定客户端代理到的 POP3 后端所支持的扩展 (如果这些扩展与认证后使用的命令相关, 当 Angie 透明地将客户端连接代理到后端时)。

## 代理

### 指令

#### proxy\_buffer

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>proxy_buffer size;</code>  |
| 默认值 | <code>proxy_buffer 4k 8k;</code> |
| 上下文 | mail, server                     |

设置用于代理的缓冲区大小。默认情况下, 缓冲区大小等于一个内存页。根据平台, 大小为 4K 或 8K。

#### proxy\_pass\_error\_message

|     |                                                 |
|-----|-------------------------------------------------|
| 语法  | <code>proxy_pass_error_message on   off;</code> |
| 默认值 | <code>proxy_pass_error_message off;</code>      |
| 上下文 | mail, server                                    |

指示是否将后端身份验证时获得的错误消息传递给客户端。

通常, 如果 Angie 的身份验证成功, 后端不能返回错误。如果仍然返回错误, 则意味着发生了一些内部错误。在这种情况下, 后端消息可能包含不应显示给客户端的信息。然而, 对于某些 POP3 服务器来说, 正确密码返回错误是一种正常行为。在这种情况下应该启用该指令。

### proxy\_protocol

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>proxy_protocol on   off;</code> |
| 默认值 | <code>proxy_protocol off;</code>      |
| 上下文 | mail, server                          |

为连接后端启用 PROXY 协议。

### proxy\_smtp\_auth

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>proxy_smtp_auth on   off;</code> |
| 默认值 | <code>proxy_smtp_auth off;</code>      |
| 上下文 | mail, server                           |

启用或禁用使用 `AUTH` 命令在 SMTP 后端进行用户身份验证。

如果 `XCLIENT` 也启用, 则 `XCLIENT` 命令将不会发送 `LOGIN` 参数。

### proxy\_timeout

|     |                                  |
|-----|----------------------------------|
| 语法  | <code>proxy_timeout time;</code> |
| 默认值 | <code>proxy_timeout 24h;</code>  |
| 上下文 | mail, server                     |

设置客户端或代理服务器连接上两次连续读或写操作之间的超时时间。如果在此时间内没有数据传输, 则关闭连接。

### xclient

|     |                                |
|-----|--------------------------------|
| 语法  | <code>xclient on   off;</code> |
| 默认值 | <code>xclient on;</code>       |
| 上下文 | mail, server                   |

启用或禁用在连接 SMTP 后端时传递带有客户端参数的 `XCLIENT` 命令。

使用 `XCLIENT`, MTA 能够将客户端信息写入日志并基于此数据应用各种限制。

如果启用了 `XCLIENT`, 则 Angie 在连接后端时传递以下命令:

- `EHLO` 带有服务器名称

- XCLIENT
- EHLO 或 HELO, 由客户端传递

如果客户端 IP 地址解析出的名称指向同一地址, 则在 XCLIENT 命令的 NAME 参数中传递该名称。如果找不到名称、指向不同地址, 或未指定解析器, 则在 NAME 参数中传递 [UNAVAILABLE]。如果在解析过程中发生错误, 则使用 [TEMPUNAVAIL] 值。

如果禁用了 XCLIENT, Angie 在连接后端时传递带有服务器名称的 EHLO 命令 (如果客户端传递了 EHLO), 否则传递 HELO 及服务器名称。

## RealIP

该模块用于将客户端地址和端口更改为在 PROXY 协议头中发送的地址和端口。必须通过在 `listen` 指令中设置 `proxy_protocol` 参数来先前启用 PROXY 协议。

## 配置示例

```
listen 110 proxy_protocol;  
  
set_real_ip_from 192.168.1.0/24;  
set_real_ip_from 192.168.2.1;  
set_real_ip_from 2001:0db8::/32;
```

## 指令

### set\_real\_ip\_from

|         |                                                        |
|---------|--------------------------------------------------------|
| Syntax  | <code>set_real_ip_from address   CIDR   unix::;</code> |
| Default | —                                                      |
| Context | mail, server                                           |

定义已知发送正确替换地址的受信任地址。如果指定了特殊值 `unix::`, 则所有 UNIX 域套接字都将被信任。

## SMTP

### 指令

## smtp\_auth

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>smtp_auth method ...;</code>  |
| 默认  | <code>smtp_auth plain login;</code> |
| 上下文 | mail, server                        |

设置 SMTP 客户端允许的 SASL 认证方法。支持的方法有：

|          |                                      |
|----------|--------------------------------------|
| plain    | AUTH PLAIN                           |
| login    | AUTH LOGIN                           |
| cram-md5 | AUTH CRAM-MD5。为了使该方法有效，密码必须以未加密形式存储。 |
| external | AUTH EXTERNAL                        |
| none     | 不需要认证                                |

明文认证方法（AUTH PLAIN 和 AUTH LOGIN）始终启用，但如果未指定 plain 和 login 方法，AUTH PLAIN 和 AUTH LOGIN 将不会自动包含在 `smtp_capabilities` 中。

## smtp\_capabilities

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>smtp_capabilities extension ...;</code> |
| 默认  | —                                             |
| 上下文 | mail, server                                  |

设置在响应 EHLO 命令时传递给客户端的 SMTP 协议扩展列表。根据 `starttls` 指令的值，`smtp_auth` 指令中指定的认证方法和 STARTTLS 会自动添加到此列表中。

建议指定客户端代理到的 MTA 支持的扩展（如果这些扩展与认证后使用的命令有关，当 Angie 透明代理客户端连接到后端时）。

## smtp\_client\_buffer

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>smtp_client_buffer size;</code>  |
| 默认  | <code>smtp_client_buffer 4k 8k;</code> |
| 上下文 | mail, server                           |

设置用于读取 SMTP 命令的缓冲区大小。默认情况下，缓冲区大小等于一个内存页。这是 4K 或 8K，取决于平台。

## smtp\_greeting\_delay

|     |                                   |
|-----|-----------------------------------|
| 语法  | smtp_greeting_delay <i>time</i> ; |
| 默认  | smtp_greeting_delay 0;            |
| 上下文 | mail, server                      |

允许设置在发送 SMTP 问候语之前的延迟, 以便拒绝那些未等候问候语即发送 SMTP 命令的客户端。

## SSL

提供了邮件代理服务器使用 SSL/TLS 协议所需的支持。

当从源代码 构建时, 默认不会构建此模块; 需要通过 `--with-mail_ssl_module` 构建选项启用。

在来自 我们仓库的软件包和镜像中, 此模块已包含在构建中。

### 重要

此模块需要 OpenSSL 库。

## 配置示例

为了减少处理器负载, 建议

- 设置工作进程 的数量等于处理器的数量,
- 启用共享 会话缓存,
- 禁用内置 会话缓存,
- 并可能增加会话生命周期 (默认 5 分钟):

```
mail {  
  
    ...  
  
    server {  
        listen          993 ssl;  
  
        ssl_protocols   TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;  
        ssl_ciphers     AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;  
        ssl_certificate  /usr/local/angie/conf/cert.pem;  
        ssl_certificate_key /usr/local/angie/conf/cert.key;  
        ssl_session_cache shared:SSL:10m;  
        ssl_session_timeout 10m;  
    }  
}
```

```
# ...  
}
```

## 指令

### ssl\_certificate

|     |                               |
|-----|-------------------------------|
| 语法  | ssl_certificate <i>file</i> ; |
| 默认值 | —                             |
| 上下文 | mail, server                  |

指定给定服务器的 PEM 格式证书文件。如果除了主证书外还需要指定中间证书, 则应该按照以下顺序在同一文件中指定: 首先是主证书, 然后是中间证书。PEM 格式的私钥可以放在同一文件中。

可以多次指定此指令以加载不同类型的证书, 例如 RSA 和 ECDSA:

```
server {  
    listen          993 ssl;  
  
    ssl_certificate  example.com.rsa.crt;  
    ssl_certificate_key example.com.rsa.key;  
  
    ssl_certificate  example.com.ecdsa.crt;  
    ssl_certificate_key example.com.ecdsa.key;  
  
    # ...  
}
```

只有 OpenSSL 1.0.2 或更高版本支持为不同的证书链设置单独的证书链。在旧版本中, 只能使用一个证书链。

#### 重要

使用 OpenSSL 1.0.2 或更高版本时, 文件名中可以使用变量:

```
ssl_certificate      $ssl_server_name.crt;  
ssl_certificate_key $ssl_server_name.key;
```

请注意, 使用变量意味着每次 SSL 握手都会加载证书, 这可能会对性能产生负面影响。

可以指定“data:certificate”作为 file 的值, 这将加载不使用中间文件的证书。

请注意, 不当使用此语法可能会带来安全隐患, 例如将私钥数据写入错误日志。

## ssl\_certificate\_key

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>ssl_certificate_key file;</code> |
| 默认值 | —                                      |
| 上下文 | mail, server                           |

指定给定服务器的 PEM 格式私钥文件。

### 重要

使用 OpenSSL 1.0.2 或更高版本时, 文件名中可以使用变量。

可以指定“engine:name:id”作为 `file` 的值, 从 OpenSSL 引擎 `name` 中加载指定 `id` 的私钥。

可以指定“data:key”作为 `file` 的值, 这将加载不使用中间文件的私钥。请注意, 不当使用此语法可能会带来安全隐患, 例如将私钥数据写入错误日志。

## ssl\_ciphers

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>ssl_ciphers ciphers;</code>          |
| 默认值 | <code>ssl_ciphers HIGH:!aNULL:!MD5;</code> |
| 上下文 | mail, server                               |

指定启用的密码套件。密码套件的格式是 OpenSSL 库理解的格式, 例如:

```
ssl_ciphers ALL:!aNULL:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
```

可以使用“`openssl ciphers`”命令查看完整列表。

## ssl\_client\_certificate

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>ssl_client_certificate file;</code> |
| 默认值 | —                                         |
| 上下文 | mail, server                              |

指定用于验证客户端证书的 PEM 格式受信任 CA 证书文件。

证书列表将发送给客户端。如果不希望这样, 可以使用 `ssl_trusted_certificate` 指令。

## ssl\_conf\_command

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>ssl_conf_command name value;</code> |
| 默认值 | —                                         |
| 上下文 | mail, server                              |

设置任意 OpenSSL 配置 命令。

### 重要

使用 OpenSSL 1.0.2 或更高版本时支持该指令。

可以在同一级别指定多个 `ssl_conf_command` 指令：

```
ssl_conf_command Options PrioritizeChaCha;  
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
```

这些指令仅在当前级别没有定义 `ssl_conf_command` 指令时，从上一级配置继承。

### 小心

请注意，直接配置 OpenSSL 可能会导致意外行为。

## ssl\_crl

|     |                            |
|-----|----------------------------|
| 语法  | <code>ssl_crl file;</code> |
| 默认值 | —                          |
| 上下文 | mail, server               |

指定用于验证 客户端证书的 PEM 格式吊销证书 (CRL) 文件。

## ssl\_dhparam

|     |                                |
|-----|--------------------------------|
| 语法  | <code>ssl_dhparam file;</code> |
| 默认值 | —                              |
| 上下文 | mail, server                   |

指定用于 DHE 密码套件的 DH 参数文件。



**小心**

默认情况下未设置参数, 因此不会使用 DHE 密码套件。

### ssl\_ecdh\_curve

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>ssl_ecdh_curve curve;</code> |
| 默认值 | <code>ssl_ecdh_curve auto;</code>  |
| 上下文 | mail, server                       |

指定用于 ECDHE 密码套件的曲线。

**重要**

使用 OpenSSL 1.0.2 或更高版本时, 可以指定多个曲线, 例如:

```
ssl_ecdh_curve prime256v1:secp384r1;
```

特殊值 auto 指示 Angie 在使用 OpenSSL 1.0.2 或更高版本时使用 OpenSSL 库内置的列表, 或在旧版本中使用 prime256v1。

**重要**

使用 OpenSSL 1.0.2 或更高版本时, 该指令设置服务器支持的曲线列表。因此, 为了使 ECDSA 证书正常工作, 重要的是要包括证书中使用的曲线。

### ssl\_password\_file

|     |                                      |
|-----|--------------------------------------|
| 语法  | <code>ssl_password_file file;</code> |
| 默认值 | —                                    |
| 上下文 | mail, server                         |

指定私钥 的密码短语文件, 其中每个密码短语在单独的行中指定。加载密钥时依次尝试这些密码短语。

示例:

```
mail {  
    ssl_password_file /etc/keys/global.pass;  
    ...  
}
```

```
server {
    server_name mail1.example.com;
    ssl_certificate_key /etc/keys/first.key;
}

server {
    server_name mail2.example.com;

    # 可以使用命名管道代替文件
    ssl_password_file /etc/keys/fifo;
    ssl_certificate_key /etc/keys/second.key;
}
}
```

### ssl\_prefer\_server\_ciphers

|     |                                     |
|-----|-------------------------------------|
| 语法  | ssl_prefer_server_ciphers on   off; |
| 默认值 | ssl_prefer_server_ciphers off;      |
| 上下文 | mail, server                        |

指定在使用 SSLv3 和 TLS 协议时应优先选择服务器密码套件而不是客户端密码套件。

### ssl\_protocols

|     |                                                                      |
|-----|----------------------------------------------------------------------|
| 语法  | ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2] [TLSv1.3]; |
| 默认值 | ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;                         |
| 上下文 | mail, server                                                         |

在 1.2.0 版本发生变更: TLSv1.3 参数已添加到默认设置中。

启用指定的协议。

**📌 重要**

TLSv1.1 和 TLSv1.2 参数仅在使用 OpenSSL 1.0.1 或更高版本时有效。

TLSv1.3 参数仅在使用 OpenSSL 1.1.1 或更高版本时有效。

## ssl\_session\_cache

|     |                                                                                  |
|-----|----------------------------------------------------------------------------------|
| 语法  | <code>ssl_session_cache off   none   [builtin[:size]] [shared:name:size];</code> |
| 默认值 | <code>ssl_session_cache none;</code>                                             |
| 上下文 | mail, server                                                                     |

设置存储会话参数的缓存类型和大小。缓存可以是以下任意类型：

|         |                                                                                                                                                           |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| off     | 严格禁止使用会话缓存：Angie 明确告知客户端会话不能被重用。                                                                                                                          |
| none    | 温和地不允许使用会话缓存：Angie 告知客户端会话可以被重用，但实际上并不在缓存中存储会话参数。                                                                                                         |
| builtin | OpenSSL 内置缓存；仅供一个工作进程使用。缓存大小以会话数量指定。如果未给出大小，则等于 20480 个会话。使用内置缓存可能导致内存碎片化。                                                                                |
| shared  | 所有工作进程共享的缓存。缓存大小以字节指定；一兆字节可以存储约 4000 个会话。每个共享缓存应有一个任意名称。可以在多个服务器中使用具有相同名称的缓存。它还用于自动生成、存储和定期轮换 TLS 会话票证密钥，除非使用 <code>ssl_session_ticket_key</code> 指令显式配置。 |

这两种缓存类型可以同时使用，例如：

```
ssl_session_cache builtin:1000 shared:SSL:10m;
```

但仅使用共享缓存而不使用内置缓存的效率可能更高。

## ssl\_session\_ticket\_key

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>ssl_session_ticket_key file;</code> |
| 默认值 | —                                         |
| 上下文 | mail, server                              |

设置用于加密和解密 TLS 会话票据的密钥文件。该指令在多个服务器需要共享同一个密钥时是必要的。默认情况下，使用随机生成的密钥。

如果指定了多个密钥，只有第一个密钥用于加密 TLS 会话票据。这允许配置密钥轮换，例如：

```
ssl_session_ticket_key current.key;  
ssl_session_ticket_key previous.key;
```

文件必须包含 80 或 48 字节的随机数据，可以使用以下命令创建：

```
openssl rand 80 > ticket.key
```

根据文件大小，使用 AES256（对于 80 字节密钥）或 AES128（对于 48 字节密钥）进行加密。

### ssl\_session\_tickets

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>ssl_session_tickets on   off;</code> |
| 默认值 | <code>ssl_session_tickets on;</code>       |
| 上下文 | mail, server                               |

启用或禁用通过 TLS 会话票据 的会话恢复。

### ssl\_session\_timeout

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>ssl_session_timeout time;</code> |
| 默认值 | <code>ssl_session_timeout 5m;</code>   |
| 上下文 | mail, server                           |

指定客户端可以重用会话参数的时间。

### ssl\_trusted\_certificate

|     |                                            |
|-----|--------------------------------------------|
| 语法  | <code>ssl_trusted_certificate file;</code> |
| 默认值 | —                                          |
| 上下文 | mail, server                               |

指定用于验证 客户端证书的 PEM 格式的受信任 CA 证书文件。

与通过 `ssl_client_certificate` 设置的证书不同, 这些证书列表不会发送给客户端。

### ssl\_verify\_client

|     |                                                                      |
|-----|----------------------------------------------------------------------|
| 语法  | <code>ssl_verify_client on   off   optional   optional_no_ca;</code> |
| 默认值 | <code>ssl_verify_client off;</code>                                  |
| 上下文 | mail, server                                                         |

启用客户端证书的验证。验证结果在认证 请求的 “Auth-SSL-Verify” 头中传递。

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| optional       | 请求客户端证书并在证书存在时验证。                                                   |
| optional_no_ca | 请求客户端证书但不要求其由受信任的 CA 证书签名。这适用于当一个外部服务执行实际证书验证的场合。证书内容通过请求发送 到认证服务器。 |

## ssl\_verify\_depth

|     |                                       |
|-----|---------------------------------------|
| 语法  | <code>ssl_verify_depth number;</code> |
| 默认值 | <code>ssl_verify_depth 1;</code>      |
| 上下文 | mail, server                          |

设置客户端证书链的验证深度。

## starttls

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>starttls on   off   only;</code> |
| 默认值 | <code>starttls off;</code>             |
| 上下文 | mail, server                           |

设置客户端证书链的验证深度。

|      |                                                      |
|------|------------------------------------------------------|
| on   | 允许在 POP3 中使用 STLS 命令以及在 IMAP 和 SMTP 中使用 STARTTLS 命令; |
| off  | 禁止使用 STLS 和 STARTTLS 命令;                             |
| only | 要求预先进行 TLS 转换。                                       |

核心邮件模块实现了邮件代理服务器的基本功能: 包括对 SMTP、IMAP 和 POP3 协议的支持, 配置服务器块, 邮件请求路由, 用户身份验证, 以及用于保护邮件连接的 SSL/TLS 支持。

本节中的其他模块扩展了该功能, 使您可以灵活配置和优化邮件服务器, 以满足各种场景和需求。

当从源代码构建时, 默认情况下不会构建此模块; 应使用 `--with-mail` 构建选项启用它。在来自我们的仓库的软件包和镜像中, 该模块已包含在构建中。

## 配置示例

```
worker_processes auto;

error_log /var/log/angie/error.log info;

events {
    worker_connections 1024;
}

mail {
    server_name      mail.example.com;
    auth_http       localhost:9000/cgi-bin/auth.cgi;
```

```
imap_capabilities IMAP4rev1 UIDPLUS IDLE LITERAL+ QUOTA;

pop3_auth          plain apop cram-md5;
pop3_capabilities LAST TOP USER PIPELINING UIDL;

smtp_auth          login plain cram-md5;
smtp_capabilities "SIZE 10485760" ENHANCEDSTATUSCODES 8BITMIME DSN;
xclient            off;

server {
    listen 25;
    protocol smtp;
}
server {
    listen 110;
    protocol pop3;
    proxy_pass_error_message on;
}
server {
    listen 143;
    protocol imap;
}
server {
    listen 587;
    protocol smtp;
}
}
```

## 指令

### listen

|     |                                                                                                                                                                                                      |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 语法  | <code>listen address[:port] [ssl] [proxy_protocol] [backlog=number] [rcvbuf=size] [sndbuf=size] [bind] [ipv6only=on   off] [reuseport] [so_keepalive=on off][keepidle]:[keepintvl]:[keepcnt];</code> |
| 默认值 | —                                                                                                                                                                                                    |
| 上下文 | server                                                                                                                                                                                               |

设置服务器接受请求的套接字的 *address* 和 *port*。可以只指定端口。地址也可以是主机名，例如：

```
listen 127.0.0.1:110;
listen *:110;
listen 110;      # 与 *:110 相同
listen localhost:110;
```

IPv6 地址用方括号指定:

```
listen [::1]:110;
listen [::]:110;
```

UNIX 域套接字以 `unix:` 前缀指定:

```
listen unix:/var/run/angie.sock;
```

### 重要

不同的服务器必须监听不同的 `address:port` 对。

|                             |                                                            |
|-----------------------------|------------------------------------------------------------|
| <code>ssl</code>            | 允许指定在此端口接受的所有连接都应以 SSL 模式工作。                               |
| <code>proxy_protocol</code> | 允许指定在此端口接受的所有连接都应使用 PROXY 协议。获取的信息传递给身份验证服务器, 并可用于更改客户端地址。 |

`listen` 指令可以有多个特定于套接字相关系统调用的附加参数。

|                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>backlog=number</code>    | 设置 <code>listen()</code> 调用中的 <code>backlog</code> 参数, 该参数限制挂起连接队列的最大长度。默认情况下, <code>backlog</code> 在 FreeBSD、DragonFly BSD 和 macOS 上设置为 -1, 在其他平台上设置为 511。                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>rcvbuf=size</code>       | 为监听套接字设置接收缓冲区大小 ( <code>SO_RCVBUF</code> 选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <code>sndbuf=size</code>       | 为监听套接字设置发送缓冲区大小 ( <code>SO_SNDBUF</code> 选项)。                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <code>bind</code>              | 此参数指示为给定的 <code>address:port</code> 对进行单独的 <code>bind()</code> 调用。事实上, 如果有多个具有相同 <code>port</code> 但不同地址的 <code>listen</code> 指令, 而其中一个 <code>listen</code> 指令监听给定端口的所有地址 ( <code>*:port</code> ), Angie 将只对 <code>*:port</code> 进行 <code>bind()</code> 。需要注意的是, 在这种情况下, 将进行 <code>getsockname()</code> 系统调用以确定接受连接的地址。如果使用了 <code>backlog</code> 、 <code>rcvbuf</code> 、 <code>sndbuf</code> 、 <code>ipv6only</code> 、 <code>reuseport</code> 或 <code>so_keepalive</code> 参数, 则对于给定的 <code>address:port</code> 对, 将始终进行单独的 <code>bind()</code> 调用。 |
| <code>ipv6only=on   off</code> | 此参数 (通过 <code>IPV6_V6ONLY</code> 套接字选项) 确定监听通配符地址 <code>[::]</code> 的 IPv6 套接字仅接受 IPv6 连接还是同时接受 IPv6 和 IPv4 连接。默认情况下, 此参数为开启状态。启动时只能设置一次。                                                                                                                                                                                                                                                                                                                                                                                                           |

`so_keepalive=on | off | [keepidle]:[keepintvl]:[keepcnt]` 为监听套接字配置“TCP keepalive”行为。

|                  |                                     |
|------------------|-------------------------------------|
| <code>''</code>  | 如果省略此参数, 则套接字将使用操作系统的设置             |
| <code>on</code>  | 为套接字启用 <code>SO_KEEPALIVE</code> 选项 |
| <code>off</code> | 为套接字禁用 <code>SO_KEEPALIVE</code> 选项 |

某些操作系统支持使用 `TCP_KEEPIDL`、`TCP_KEEPIVL` 和 `TCP_KEEPCNT` 套接字选项在每个套接字基础

上设置 TCP keepalive 参数。在这些系统上（目前为 Linux 2.4+、NetBSD 5+ 和 FreeBSD 9.0-STABLE），可以使用 `keepidle`、`keepintvl` 和 `keepcnt` 参数进行配置。可以省略一个或两个参数，在这种情况下，将使用系统默认设置。

例如，

```
so_keepalive=30m::10
```

将空闲超时（`TCP_KEEPIDLE`）设置为 30 分钟，保持探测间隔（`TCP_KEEPINTVL`）为系统默认值，将探测次数（`TCP_KEEPCNT`）设置为 10 次探测。

### mail

|     |                           |
|-----|---------------------------|
| 语法  | <code>mail { ... }</code> |
| 默认值 | —                         |
| 上下文 | main                      |

提供指定邮件服务器指令的配置文件上下文。

### max\_commands

Added in version 1.7.0.

|     |                                   |
|-----|-----------------------------------|
| 语法  | <code>max_commands number;</code> |
| 默认值 | <code>max_commands 1000;</code>   |
| 上下文 | mail, server                      |

设置在身份验证期间发出的命令的最大数量，以增强对 DoS 攻击的防护。

### max\_errors

|     |                                 |
|-----|---------------------------------|
| 语法  | <code>max_errors number;</code> |
| 默认值 | <code>max_errors 5;</code>      |
| 上下文 | mail, server                    |

设置超过此错误数后将关闭连接的协议错误数量。



## protocol

|     |                                           |
|-----|-------------------------------------------|
| 语法  | <code>protocol imap   pop3   smtp;</code> |
| 默认值 | —                                         |
| 上下文 | server                                    |

设置代理服务器的协议。支持的协议有 *IMAP*、*POP3* 和 *SMTP*。

如果未设置该指令，可以根据 *listen* 指令中指定的知名端口自动检测协议：

```
imap: 143, 993
pop3: 110, 995
smtp: 25, 587, 465
```

当从源代码构建时，可以使用 `--without-mail_imap_module`、`--without-mail_pop3_module` 和 `--without-mail_smtp_module` 构建选项禁用不必要的协议。

## resolver

|     |                                                                                                    |
|-----|----------------------------------------------------------------------------------------------------|
| 语法  | <code>resolver address ... [valid=time] [ipv4=on   off] [ipv6=on   off] [status_zone=zone];</code> |
| 默认值 | <code>resolver off;</code>                                                                         |
| 上下文 | mail, server                                                                                       |

配置用于查找客户端主机名的名称服务器，以将其传递给身份验证服务器，以及在代理 SMTP 时在 *XCLIENT* 命令中使用。例如：

```
resolver 127.0.0.53 [::1]:5353;
```

地址可以指定为域名或 IP 地址，并且可以指定可选端口。如果未指定端口，则使用端口 53。名称服务器以轮循方式查询。

默认情况下，Angie 使用响应的 TTL 值缓存答案。可选的 `valid` 参数允许覆盖它：

|                    |                                     |
|--------------------|-------------------------------------|
| <code>valid</code> | 可选 <code>valid</code> 参数允许覆盖缓存条目有效性 |
|--------------------|-------------------------------------|

```
resolver 127.0.0.53 [::1]:5353 valid=30s;
```

默认情况下，Angie 在解析时会查找 IPv4 和 IPv6 地址。

|                          |                     |
|--------------------------|---------------------|
| <code>ipv4=off</code>    | 禁用查找 IPv4 地址        |
| <code>ipv6=off</code>    | 禁用查找 IPv6 地址        |
| <code>status_zone</code> | 可选参数, 为指定区域启用统计数据收集 |

### 💡 小技巧

为防止 DNS 欺骗, 建议在适当安全的受信任本地网络中配置 DNS 服务器。

## resolver\_timeout

|     |                                     |
|-----|-------------------------------------|
| 语法  | <code>resolver_timeout time;</code> |
| 默认值 | <code>resolver_timeout 30s;</code>  |
| 上下文 | mail, server                        |

为 DNS 操作设置超时, 例如:

```
resolver_timeout 5s;
```

## server

|     |                             |
|-----|-----------------------------|
| 语法  | <code>server { ... }</code> |
| 默认值 | —                           |
| 上下文 | mail                        |

设置服务器的配置。

## server\_name

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>server_name name;</code>     |
| 默认值 | <code>server_name hostname;</code> |
| 上下文 | mail, server                       |

设置使用的服务器名称:

- 在初始 POP3/SMTP 服务器问候中;
- 在 SASL CRAM-MD5 身份验证期间作为盐;

- 在连接到 SMTP 后端时, 在 EHLO 命令中使用, 如果启用了传递 *XCLIENT* 命令。

如果未指定该指令, 则使用机器的 *hostname*。

## timeout

|     |                            |
|-----|----------------------------|
| 语法  | <code>timeout time;</code> |
| 默认值 | <code>timeout 60s;</code>  |
| 上下文 | mail, server               |

设置在开始代理到后端之前使用的超时时间。

### 3.3.5 Google PerfTools 模块

该模块使用 Google 性能工具 启用 Angie 工作进程的性能分析。该模块旨在为 Angie 开发人员提供服务, 允许他们通过提供有关内存使用、CPU 使用及其他性能相关指标的详细见解来分析和优化服务器性能。

当从源代码 构建时, 该模块默认不被构建; 需通过使用 `--with-google_perftools_module` 构建选项来启用。

#### 重要

该模块需要 `gperftools` 库。

## 配置示例

```
google_perftools_profiles /var/log/angie/perftools;
```

配置文件将存储为 `/var/log/angie/perftools.<worker PID>` 文件。

## 指令

### google\_perftools\_profiles

|     |                                               |
|-----|-----------------------------------------------|
| 语法  | <code>google_perftools_profiles 文件名前缀;</code> |
| 默认值 | —                                             |
| 上下文 | main                                          |

设置用于存储 Angie 工作进程性能分析信息的文件名前缀。工作进程 ID 将附加在名称末尾, 并以点号分隔, 例如: `/var/log/angie/perftools.1234`。

### 3.3.6 WASM 模块

#### WAMR

该模块支持与 WebAssembly Micro Runtime 集成以执行 WASM 代码, 为 `wasm_modules` 上下文添加了许多特定于运行时的指令。

在我们的代码库中, 模块是动态构建的, 并作为一个名为 `angie-module-wamr` 的独立包提供。

#### 配置示例

```
wasm_modules {  
  
    wamr_heap_size 16k;  
  
    wamr_stack_size 16k;  
  
    load fft_transform.wasm id=fft;  
}
```

#### 指令

##### wamr\_heap\_size

|     |                                   |
|-----|-----------------------------------|
| 语法  | <code>wamr_heap_size size;</code> |
| 默认值 | <code>wamr_heap_size 8k;</code>   |
| 上下文 | <code>wasm_modules</code>         |

设置单个模块实例的堆大小。

##### wamr\_global\_heap\_size

|     |                                          |
|-----|------------------------------------------|
| 语法  | <code>wamr_global_heap_size size;</code> |
| 默认值 | <code>wamr_global_heap_size 1m;</code>   |
| 上下文 | <code>wasm_modules</code>                |

设置整个 WAMR 运行时的堆大小。

## wamr\_stack\_size

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>wamr_stack_size size;</code> |
| 默认值 | <code>wamr_stack_size 8k;</code>   |
| 上下文 | <code>wasm_modules</code>          |

设置单个模块实例的栈大小。

## Wasmtime

该模块支持与 `Wasmtime` 运行时集成, 以执行 WASM 代码, 并为 `wasm_modules` 上下文添加一些运行时特定的指令。

在我们的仓库中, 该模块是动态构建的, 并作为一个名为 `angie-module-wasmtime` 的独立软件包提供。

## 配置示例

```
wasm_modules {  
  
    wasmtime_stack_size 8k;  
  
    wasmtime_enable_wasi on;  
  
    load fft_transform.wasm id=fft;  
}
```

## 指令

### wasmtime\_enable\_wasi

|     |                                             |
|-----|---------------------------------------------|
| 语法  | <code>wasmtime_enable_wasi on   off;</code> |
| 默认值 | <code>wasmtime_enable_wasi on;</code>       |
| 上下文 | <code>wasm_modules</code>                   |

启用或禁用 `WebAssembly System Interface` APIs, 这些接口为在 `Angie` 上运行的 WASM 模块提供基本的类 `POSIX` 功能。

#### 备注

可以使用 `load` 指令将 `Angie` 特定的 API 加入白名单。

## wasmtime\_stack\_size

|     |                                        |
|-----|----------------------------------------|
| 语法  | <code>wasmtime_stack_size size;</code> |
| 默认值 | <code>wasmtime_stack_size 8k;</code>   |
| 上下文 | <code>wasm_modules</code>              |

设置 `max_wasm_stack` 值为指定的 `size`, 从而限制执行 WASM 代码时可用的最大堆栈空间。

此核心模块在 Angie 中实现了基本的 WASM 功能: 包括支持加载替代运行时和 WASM 模块, 以及配置它们的功能和限制。

本节中的其他模块扩展了此功能, 使您能够灵活地配置和优化 WASM 功能, 以满足各种场景和需求。

在我们的代码库中, 该模块是动态构建的, 并作为单独的软件包提供, 名为 `angie-module-wasm`。

### 配置示例

```
# 这些指令加载核心功能
load_module modules/nginx_wasm_module.so;
load_module modules/nginx_wasm_core_module.so;

load_module modules/nginx_wasmtime_module.so;

# 可在此处获得: https://git.angie.software/web-server/angie-wasm
load_module modules/nginx_http_wasm_host_module.so;
load_module modules/nginx_http_wasm_content_module.so;
load_module modules/nginx_http_wasm_vars_module.so;

events {
}

wasm_modules {
    #use wasmtime;

    load ngx_http_handler.wasm id=handler;
    load ngx_http_vars.wasm id=vars type=reactor;
}

http {
    wasm_var vars "ngx:wasi/var-utils#sum-entry" $rvar $arg_a $arg_b $arg_c $arg_d;

    server {

        listen *:8080;
```

```
location / {  
  
    return 200 "sum('$arg_a','$arg_b','$arg_c','$arg_d')=$rvar\n";  
}  
  
location /wasm {  
  
    client_max_body_size 20M;  
    wasm_content handler "ngx:wasi/http-handler-entry#handle-request";  
}  
}
```

## 指令

### load

|     |                                                                                                 |
|-----|-------------------------------------------------------------------------------------------------|
| 语法  | <code>load file id=id [fs=host_path:guest_path]... [api=api]... [type=command   reactor]</code> |
| 默认  | —                                                                                               |
| 上下文 | wasm_modules                                                                                    |

从磁盘 *file* 加载模块并为其分配一个 *id* (必需)。在加载过程中, 模块会被验证以确保可以实例化。

该指令具有以下参数:

|      |                                                                                                                                                                                                      |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fs   | 允许来宾访问主机上的目录。可以为不同的目录多次指定。                                                                                                                                                                           |
| api  | 启用模块可用 API 的白名单模式, 并指定这些 API。如果模块尝试使用受限 API (即此处未列出), 将返回“找不到 API”错误。<br>默认情况下, 模块可以访问所有 API。                                                                                                        |
| type | 控制加载模块的生命周期。 <ul style="list-style-type: none"><li>在 <code>command</code> 模式下, 机器运行一次, 执行后其状态被销毁。</li><li>在 <code>reactor</code> 模式下, 机器实际上无限期运行, 允许多次代码执行。这需要谨慎的内存管理: 如果资源未清理, 可能会导致内存泄漏。</li></ul> |

## wasm\_modules

|     |                                    |
|-----|------------------------------------|
| 语法  | <code>wasm_modules { ... };</code> |
| 默认  | —                                  |
| 上下文 | main                               |

一个顶级块指令，提供配置文件上下文，其中应指定 WASM 指令。它可以包含加载 WASM 模块和配置特定运行时设置的指令。

### 3.3.7 核心模块

*Core* 负责管理服务文件、进程和其他 Angie 模块。

### 3.3.8 HTTP 模块

|                      |                                                        |
|----------------------|--------------------------------------------------------|
| <i>HTTP</i>          | 处理 HTTP 请求和响应的核心功能，管理 HTTP 服务器、连接和静态文件。                |
| <i>Access</i>        | 基于 IP 和 CIDR 范围的访问控制。                                  |
| <i>ACME</i>          | 使用 ACME 协议自动获取 SSL 证书。                                 |
| <i>Addition</i>      | 在响应体之前或之后插入预定义的片段。                                     |
| <i>API</i>           | RESTful HTTP 接口，用于获取关于 Web 服务器的基本信息及其统计数据，以 JSON 格式呈现。 |
| <i>Auth Basic</i>    | 基于用户名和密码的基本 HTTP 认证访问控制。                               |
| <i>Auth Request</i>  | 使用对外部 HTTP 服务的子请求进行授权。                                 |
| <i>AutoIndex</i>     | 在没有索引文件的情况下自动生成目录列表。                                   |
| <i>Browser</i> (已弃用) | 基于 <i>User-Agent</i> 头的浏览器识别。                          |
| <i>Charset</i>       | 响应编码的配置和转换。                                            |
| <i>DAV</i>           | 使用 WebDAV 协议在服务器上进行文件管理。                               |
| <i>Empty GIF</i>     | 提供一个像素的透明 GIF。                                         |
| <i>FastCGI</i>       | 转发请求到 FastCGI 服务器。                                     |
| <i>FLV</i>           | Flash Video (FLV) 文件的伪流。                               |
| <i>Geo</i>           | 将 IP 地址转换为预定义变量值。                                      |
| <i>GeoIP</i>         | 使用 MaxMind GeoIP 数据库通过地理定位获取 IP 地址数据。                  |
| <i>gRPC</i>          | 转发请求到 gRPC 服务器。                                        |
| <i>GunZIP</i>        | 解压缩压缩的 GZip 响应以进行修改或在客户端不支持压缩时使用。                      |
| <i>GZip</i>          | 使用 GZip 方法压缩响应以节省流量。                                   |
| <i>GZip Static</i>   | 提供已使用 GZip 方法预压缩的静态文件。                                 |
| <i>Headers</i>       | 修改响应头字段。                                               |
| <i>HTTP2</i>         | 使用 HTTP/2 协议处理请求。                                      |
| <i>HTTP3</i>         | 使用 HTTP/3 协议处理请求。                                      |
| <i>Image Filter</i>  | 图像转换。                                                  |



表 1 – 接上页

|                          |                                                               |
|--------------------------|---------------------------------------------------------------|
| <i>Index</i>             | 配置索引文件以处理带有斜杠结尾 (/) 的请求。                                      |
| <i>JS</i>                | 通过在 <i>njs</i> 中实现附加逻辑来扩展功能, <i>njs</i> 是 JavaScript 语言的一个子集。 |
| <i>Limit Conn</i>        | 限制并发请求数量 (活动连接数) 以防止过载。                                       |
| <i>Limit Req</i>         | 限制请求速率以防止过载和密码猜测。                                             |
| <i>Log</i>               | 配置请求日志以跟踪资源访问, 用于监控和分析。                                       |
| <i>Map</i>               | 基于预定义的键值对转换变量。                                                |
| <i>Memcached</i>         | 从 Memcached 服务器获取响应。                                          |
| <i>Mirror</i>            | 将请求镜像到其他服务器。                                                  |
| <i>MP4</i>               | MP4 文件的伪流。                                                    |
| <i>Perl</i>              | 通过在 Perl 语言中实现附加逻辑来扩展功能。                                      |
| <i>Prometheus</i>        | 以 Prometheus 兼容格式提供服务器指标, 用于监控和统计收集。                          |
| <i>Proxy</i>             | 反向代理请求到其他 HTTP 服务器。                                           |
| <i>Random Index</i>      | 为带有斜杠结尾 (/) 的请求随机选择索引文件。                                      |
| <i>RealIP</i>            | 在另一个代理服务器后运行时识别客户端地址和端口。                                      |
| <i>Referer</i>           | 验证 <i>Referer</i> 头的值。                                        |
| <i>Rewrite</i>           | 条件 URI 修改、重定向、变量设置和配置选择。                                      |
| <i>SCGI</i>              | 转发请求到 SCGI 服务器。                                               |
| <i>Secure Link</i>       | 创建安全链接, 并具有限制访问时间的能力。                                         |
| <i>Slice</i>             | 将请求分割为多个子请求以更好地缓存大响应。                                         |
| <i>Split Clients</i>     | 为 A/B 测试、金丝雀发布、分片和其他需要按比例分组的场景创建变量。                           |
| <i>SSI</i>               | 在响应中处理 SSI (服务器端包含) 命令。                                       |
| <i>SSL</i>               | SSL/TLS 配置以处理 HTTPS 请求。                                       |
| <i>Stub Status</i> (已弃用) | 以文本格式提供全局连接和请求计数。                                             |
| <i>Sub</i>               | 在服务器响应中搜索和替换片段。                                               |
| <i>Upstream</i>          | 配置代理服务器组以进行负载均衡。                                              |
| <i>Upstream Probe</i>    | 为代理服务器组配置健康探测。                                                |
| <i>UserID</i>            | 发放和处理唯一的客户端标识符 Cookie, 用于会话跟踪和分析。                             |
| <i>uWSGI</i>             | 转发请求到 uWSGI 服务器。                                              |
| <i>XSLT</i>              | 使用 XSLT 样式表进行 XML 转换。                                         |

### 3.3.9 流模块

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <i>Stream</i>         | 在 L4 级别平衡 TCP 和 UDP 协议的基本流服务器功能。                      |
| <i>Access</i>         | 基于 IP 和 CIDR 范围的访问控制。                                 |
| <i>Geo</i>            | 将 IP 地址转换为预定义变量值。                                     |
| <i>GeoIP</i>          | 使用 MaxMind GeoIP 数据库通过地理定位获取 IP 地址数据。                 |
| <i>JS</i>             | 通过在 njs 中实现附加逻辑来扩展功能, njs 是 JavaScript 语言的一个子集。       |
| <i>Limit Conn</i>     | 限制并发请求数量 (活动连接数) 以防止过载。                               |
| <i>Log</i>            | 配置请求日志以跟踪资源访问, 用于监控和分析。                               |
| <i>Map</i>            | 基于预定义的键值对转换变量。                                        |
| <i>MQTT Pre-read</i>  | 在做出负载均衡决策之前, 从 MQTT 连接中读取客户端标识符和用户名。                  |
| <i>Pass</i>           | 配置直接将接受的连接传递到配置的监听套接字。                                |
| <i>Proxy</i>          | 配置到其他服务器的代理。                                          |
| <i>RDP Preread</i>    | 在做出负载均衡决策之前, 从 RDP 连接中读取 Cookie。                      |
| <i>RealIP</i>         | 在另一个代理服务器后运行时识别客户端地址和端口。                              |
| <i>Return</i>         | 在连接时向客户端发送指定值, 而不进行进一步代理。                             |
| <i>Set</i>            | 设置预定义变量值。                                             |
| <i>Split Clients</i>  | 为 A/B 测试、金丝雀发布、分片和其他需要按比例分组的场景创建变量。                   |
| <i>SSL</i>            | 终止 SSL/TLS 和 DTLS 协议。                                 |
| <i>SSL Preread</i>    | 在不终止 SSL/TLS 的情况下从 ClientHello 消息中提取信息, 并在做出负载均衡决策之前。 |
| <i>Upstream</i>       | 配置代理服务器组以进行负载均衡。                                      |
| <i>Upstream Probe</i> | 为代理服务器组配置健康探测。                                        |

### 3.3.10 邮件模块

|                  |                                        |
|------------------|----------------------------------------|
| <i>Mail</i>      | 基本邮件代理服务器功能。                           |
| <i>Auth HTTP</i> | 使用 HTTP 请求对外部服务器进行用户认证和服务器选择, 以进行后续代理。 |
| <i>IMAP</i>      | 支持 IMAP 协议。                            |
| <i>POP3</i>      | 支持 POP3 协议。                            |
| <i>Proxy</i>     | 配置到其他服务器的代理。                           |
| <i>RealIP</i>    | 在另一个代理服务器后运行时识别客户端地址和端口。               |
| <i>SMTP</i>      | 支持 SMTP 协议。                            |
| <i>SSL</i>       | 支持 SSL/TLS 和 StartTLS 协议。              |

### 3.3.11 Google PerfTools 模块

*Google PerfTools* 集成 Google 性能工具库, 用于应用程序分析和性能分析。

### 3.3.12 WASM 模块

*WASM* 基本 WASM 功能可以在 Angie 中运行 WASM 代码。

*WAMR* 集成 WebAssembly Micro Runtime。

*Wasmtime* 集成 Wasmtime 运行时。

## 章节 4

---

### 知识产权

---

软件产品 Angie PRO 的文档是 Web Server, LLC 的知识产权。该文档是在对软件产品 Nginx 文档进行修改（修订）后创建的。

## 索引

## A

`absolute_redirect` (*http*), 337  
`accept_mutex` (*core*), 17  
`accept_mutex_delay` (*core*), 18  
`access_log` (*http*), 142  
`access_log` (*stream*), 398  
`acme` (*http*), 31  
`acme_client` (*http*), 31  
`add_after_body` (*http*), 33  
`add_before_body` (*http*), 33  
`add_header` (*http*), 122  
`add_trailer` (*http*), 122  
`addition_types` (*http*), 34  
`aio` (*http*), 337  
`aio_write` (*http*), 339  
`alias` (*http*), 339  
`allow` (*http*), 28  
`allow` (*stream*), 383  
`ancient_browser` (*http*), 74  
`ancient_browser_value` (*http*), 75  
`api` (*http*), 34  
`api_config_files` (*http*), 36  
`auth_basic` (*http*), 70  
`auth_basic_user_file` (*http*), 70  
`auth_delay` (*http*), 340  
`auth_http`, 465  
`auth_http_header`, 465  
`auth_http_pass_client_cert`, 465  
`auth_http_timeout`, 465  
`auth_request` (*http*), 71  
`auth_request_set` (*http*), 71  
`auto_redirect` (*http*), 340  
`autoindex` (*http*), 72

`autoindex_exact_size` (*http*), 72  
`autoindex_format` (*http*), 72  
`autoindex_localtime` (*http*), 73

## B

`bind_conn` (*http*), 280  
`break` (*http*), 225

## C

`charset` (*http*), 76  
`charset_map` (*http*), 76  
`charset_types` (*http*), 77  
`chunked_transfer_encoding` (*http*), 340  
`client_body_buffer_size` (*http*), 340  
`client_body_in_file_only` (*http*), 341  
`client_body_in_single_buffer` (*http*), 341  
`client_body_temp_path` (*http*), 341  
`client_body_timeout` (*http*), 342  
`client_header_buffer_size` (*http*), 342  
`client_header_timeout` (*http*), 342  
`client_max_body_size` (*http*), 343  
`connection_pool_size` (*http*), 343  
`create_full_put_path` (*http*), 79

## D

`daemon` (*core*), 18  
`dav_access` (*http*), 79  
`dav_methods` (*http*), 79  
`debug_connection` (*core*), 18  
`debug_points` (*core*), 19  
`default_type` (*http*), 343  
`deny` (*http*), 28  
`deny` (*stream*), 383  
`directio` (*http*), 344

`directio_alignment (http)`, 344  
`disable_symlinks (http)`, 344

## E

`empty_gif (http)`, 81  
`env (core)`, 19  
`error_log (core)`, 20  
`error_page (http)`, 345  
`etag (http)`, 346  
`events (core)`, 20  
`expires (http)`, 122

## F

`fastcgi_bind (http)`, 82  
`fastcgi_buffer_size (http)`, 82  
`fastcgi_buffering (http)`, 82  
`fastcgi_buffers (http)`, 83  
`fastcgi_busy_buffers_size (http)`, 83  
`fastcgi_cache (http)`, 83  
`fastcgi_cache_background_update (http)`, 83  
`fastcgi_cache_bypass (http)`, 84  
`fastcgi_cache_key (http)`, 84  
`fastcgi_cache_lock (http)`, 84  
`fastcgi_cache_lock_age (http)`, 85  
`fastcgi_cache_lock_timeout (http)`, 85  
`fastcgi_cache_max_range_offset (http)`, 85  
`fastcgi_cache_methods (http)`, 85  
`fastcgi_cache_min_uses (http)`, 86  
`fastcgi_cache_path (http)`, 86  
`fastcgi_cache_revalidate (http)`, 87  
`fastcgi_cache_use_stale (http)`, 87  
`fastcgi_cache_valid (http)`, 88  
`fastcgi_catch_stderr (http)`, 89  
`fastcgi_connect_timeout (http)`, 89  
`fastcgi_connection_drop (http)`, 90  
`fastcgi_force_ranges (http)`, 90  
`fastcgi_hide_header (http)`, 90  
`fastcgi_ignore_client_abort (http)`, 90  
`fastcgi_ignore_headers (http)`, 91  
`fastcgi_index (http)`, 91  
`fastcgi_intercept_errors (http)`, 91  
`fastcgi_keep_conn (http)`, 92  
`fastcgi_limit_rate (http)`, 92  
`fastcgi_max_temp_file_size (http)`, 92  
`fastcgi_next_upstream (http)`, 93

`fastcgi_next_upstream_timeout (http)`, 94  
`fastcgi_next_upstream_tries (http)`, 94  
`fastcgi_no_cache (http)`, 94  
`fastcgi_param (http)`, 95  
`fastcgi_pass (http)`, 96  
`fastcgi_pass_header (http)`, 96  
`fastcgi_pass_request_body (http)`, 96  
`fastcgi_pass_request_headers (http)`, 96  
`fastcgi_read_timeout (http)`, 97  
`fastcgi_request_buffering (http)`, 97  
`fastcgi_send_lowat (http)`, 97  
`fastcgi_send_timeout (http)`, 98  
`fastcgi_socket_keepalive (http)`, 98  
`fastcgi_split_path_info (http)`, 98  
`fastcgi_store (http)`, 99  
`fastcgi_store_access (http)`, 100  
`fastcgi_temp_file_write_size (http)`, 100  
`fastcgi_temp_path (http)`, 100  
`feedback (http)`, 281  
`feedback (stream)`, 441  
`flv (http)`, 102

## G

`geo (http)`, 102  
`geo (stream)`, 384  
`geoip_city (http)`, 105  
`geoip_city (stream)`, 386  
`geoip_country (http)`, 105  
`geoip_country (stream)`, 386  
`geoip_org (http)`, 105  
`geoip_org (stream)`, 387  
`geoip_proxy (http)`, 106  
`geoip_proxy_recursive (http)`, 106  
`google_perftools_profiles`, 488  
`grpc_bind (http)`, 107  
`grpc_buffer_size (http)`, 107  
`grpc_connect_timeout (http)`, 108  
`grpc_connection_drop (http)`, 108  
`grpc_hide_header (http)`, 108  
`grpc_ignore_headers (http)`, 108  
`grpc_intercept_errors (http)`, 109  
`grpc_next_upstream (http)`, 109  
`grpc_next_upstream_timeout (http)`, 110  
`grpc_next_upstream_tries (http)`, 111  
`grpc_pass (http)`, 111

[grpc\\_pass\\_header \(http\), 111](#)  
[grpc\\_read\\_timeout \(http\), 112](#)  
[grpc\\_send\\_timeout \(http\), 112](#)  
[grpc\\_set\\_header \(http\), 112](#)  
[grpc\\_socket\\_keepalive \(http\), 112](#)  
[grpc\\_ssl\\_certificate \(http\), 113](#)  
[grpc\\_ssl\\_certificate\\_key \(http\), 113](#)  
[grpc\\_ssl\\_ciphers \(http\), 113](#)  
[grpc\\_ssl\\_conf\\_command \(http\), 114](#)  
[grpc\\_ssl\\_crl \(http\), 114](#)  
[grpc\\_ssl\\_name \(http\), 114](#)  
[grpc\\_ssl\\_password\\_file \(http\), 114](#)  
[grpc\\_ssl\\_protocols \(http\), 115](#)  
[grpc\\_ssl\\_server\\_name \(http\), 115](#)  
[grpc\\_ssl\\_session\\_reuse \(http\), 115](#)  
[grpc\\_ssl\\_trusted\\_certificate \(http\), 115](#)  
[grpc\\_ssl\\_verify \(http\), 116](#)  
[grpc\\_ssl\\_verify\\_depth \(http\), 116](#)  
[gunzip \(http\), 117](#)  
[gunzip\\_buffers \(http\), 117](#)  
[gzip \(http\), 118](#)  
[gzip\\_buffers \(http\), 118](#)  
[gzip\\_comp\\_level \(http\), 118](#)  
[gzip\\_disable \(http\), 118](#)  
[gzip\\_http\\_version \(http\), 119](#)  
[gzip\\_min\\_length \(http\), 119](#)  
[gzip\\_proxied \(http\), 119](#)  
[gzip\\_static \(http\), 121](#)  
[gzip\\_types \(http\), 120](#)  
[gzip\\_vary \(http\), 120](#)

## H

[hash \(http\), 282](#)  
[hash \(stream\), 442](#)  
[http \(http\), 346](#)  
[http2 \(http\), 328](#)  
[http2\\_body\\_preread\\_size \(http\), 328](#)  
[http2\\_chunk\\_size \(http\), 328](#)  
[http2\\_max\\_concurrent\\_pushes \(http\), 328](#)  
[http2\\_max\\_concurrent\\_streams \(http\), 328](#)  
[http2\\_push \(http\), 329](#)  
[http2\\_push\\_preload \(http\), 329](#)  
[http2\\_recv\\_buffer\\_size \(http\), 329](#)  
[http3 \(http\), 331](#)  
[http3\\_hq \(http\), 331](#)

[http3\\_max\\_concurrent\\_streams \(http\), 332](#)  
[http3\\_stream\\_buffer\\_size \(http\), 332](#)

## I

[if \(http\), 225](#)  
[if\\_modified\\_since \(http\), 347](#)  
[ignore\\_invalid\\_headers \(http\), 347](#)  
[image\\_filter \(http\), 124](#)  
[image\\_filter\\_buffer \(http\), 125](#)  
[image\\_filter\\_interlace \(http\), 125](#)  
[image\\_filter\\_jpeg\\_quality \(http\), 125](#)  
[image\\_filter\\_sharpen \(http\), 126](#)  
[image\\_filter\\_transparency \(http\), 126](#)  
[image\\_filter\\_webp\\_quality \(http\), 126](#)  
[imap\\_auth, 468](#)  
[imap\\_capabilities, 469](#)  
[imap\\_client\\_buffer, 469](#)  
[include \(core\), 21](#)  
[index \(http\), 127](#)  
[internal \(http\), 347](#)  
[ip\\_hash \(http\), 283](#)

## J

[js\\_access \(stream\), 390](#)  
[js\\_body\\_filter \(http\), 130](#)  
[js\\_content \(http\), 131](#)  
[js\\_fetch\\_buffer\\_size \(http\), 131](#)  
[js\\_fetch\\_buffer\\_size \(stream\), 390](#)  
[js\\_fetch\\_ciphers \(http\), 131](#)  
[js\\_fetch\\_ciphers \(stream\), 390](#)  
[js\\_fetch\\_max\\_response\\_buffer\\_size \(http\), 131](#)  
[js\\_fetch\\_max\\_response\\_buffer\\_size \(stream\), 390](#)  
[js\\_fetch\\_protocols \(http\), 131](#)  
[js\\_fetch\\_protocols \(stream\), 391](#)  
[js\\_fetch\\_timeout \(http\), 132](#)  
[js\\_fetch\\_timeout \(stream\), 391](#)  
[js\\_fetch\\_trusted\\_certificate \(http\), 132](#)  
[js\\_fetch\\_trusted\\_certificate \(stream\), 391](#)  
[js\\_fetch\\_verify \(http\), 132](#)  
[js\\_fetch\\_verify \(stream\), 391](#)  
[js\\_fetch\\_verify\\_depth \(http\), 132](#)  
[js\\_fetch\\_verify\\_depth \(stream\), 392](#)  
[js\\_filter \(stream\), 392](#)  
[js\\_header\\_filter \(http\), 133](#)

js\_import (*http*), 133  
 js\_import (*stream*), 392  
 js\_path (*http*), 133  
 js\_path (*stream*), 393  
 js\_preload\_object (*http*), 134  
 js\_preload\_object (*stream*), 393  
 js\_preread (*stream*), 393  
 js\_set (*http*), 134  
 js\_set (*stream*), 394  
 js\_shared\_dict\_zone (*http*), 134  
 js\_shared\_dict\_zone (*stream*), 394  
 js\_var (*http*), 135  
 js\_var (*stream*), 395

## K

keepalive (*http*), 283  
 keepalive\_disable (*http*), 348  
 keepalive\_requests (*http*), 285, 348  
 keepalive\_time (*http*), 286, 349  
 keepalive\_timeout (*http*), 286, 349

## L

large\_client\_header\_buffers (*http*), 349  
 least\_conn (*http*), 286  
 least\_conn (*stream*), 442  
 least\_time (*http*), 286  
 least\_time (*stream*), 443  
 limit\_conn (*http*), 137  
 limit\_conn (*stream*), 396  
 limit\_conn\_dry\_run (*http*), 137  
 limit\_conn\_dry\_run (*stream*), 397  
 limit\_conn\_log\_level (*http*), 138  
 limit\_conn\_log\_level (*stream*), 397  
 limit\_conn\_status (*http*), 138  
 limit\_conn\_zone (*http*), 138  
 limit\_conn\_zone (*stream*), 397  
 limit\_except (*http*), 350  
 limit\_rate (*http*), 350  
 limit\_rate\_after (*http*), 351  
 limit\_req (*http*), 140  
 limit\_req\_dry\_run (*http*), 140  
 limit\_req\_log\_level (*http*), 141  
 limit\_req\_status (*http*), 141  
 limit\_req\_zone (*http*), 141  
 lingering\_close (*http*), 351

lingering\_time (*http*), 352  
 lingering\_timeout (*http*), 352  
 listen, 483  
 listen (*http*), 352  
 listen (*stream*), 453  
 load, 492  
 load\_module (*core*), 21  
 location (*http*), 356  
 lock\_file (*core*), 21  
 log\_format (*http*), 144  
 log\_format (*stream*), 399  
 log\_not\_found (*http*), 358  
 log\_subrequest (*http*), 358

## M

mail, 485  
 map (*http*), 145  
 map (*stream*), 401  
 map\_hash\_bucket\_size (*http*), 147  
 map\_hash\_bucket\_size (*stream*), 402  
 map\_hash\_max\_size (*http*), 147  
 map\_hash\_max\_size (*stream*), 402  
 master\_process (*core*), 22  
 max\_commands, 485  
 max\_errors, 485  
 max\_ranges (*http*), 359  
 memcached\_bind (*http*), 148  
 memcached\_buffer\_size (*http*), 148  
 memcached\_connect\_timeout (*http*), 149  
 memcached\_gzip\_flag (*http*), 149  
 memcached\_next\_upstream (*http*), 149  
 memcached\_next\_upstream\_timeout (*http*), 150  
 memcached\_next\_upstream\_tries (*http*), 150  
 memcached\_pass (*http*), 151  
 memcached\_read\_timeout (*http*), 151  
 memcached\_send\_timeout (*http*), 151  
 memcached\_socket\_keepalive (*http*), 151  
 merge\_slashes (*http*), 359  
 min\_delete\_depth (*http*), 80  
 mirror (*http*), 153  
 mirror\_request\_body (*http*), 153  
 modern\_browser (*http*), 75  
 modern\_browser\_value (*http*), 75  
 mp4 (*http*), 155  
 mp4\_buffer\_size (*http*), 155



- mp4\_limit\_rate (*http*), 155
  - mp4\_limit\_rate\_after (*http*), 156
  - mp4\_max\_buffer\_size (*http*), 155
  - mp4\_start\_key\_frame (*http*), 156
  - mqtt\_preread (*stream*), 403
  - msie\_padding (*http*), 359
  - msie\_refresh (*http*), 360
  - multi\_accept (*core*), 22
- O**
- open\_file\_cache (*http*), 360
  - open\_file\_cache\_errors (*http*), 361
  - open\_file\_cache\_min\_uses (*http*), 361
  - open\_file\_cache\_valid (*http*), 361
  - open\_log\_file\_cache (*http*), 144
  - open\_log\_file\_cache (*stream*), 400
  - output\_buffers (*http*), 361
  - override\_charset (*http*), 77
- P**
- pass (*stream*), 405
  - pcre\_jit (*core*), 22
  - perl (*http*), 158
  - perl\_modules (*http*), 158
  - perl\_require (*http*), 158
  - perl\_set (*http*), 159
  - pid (*core*), 23
  - pop3\_auth, 469
  - pop3\_capabilities, 470
  - port\_in\_redirect (*http*), 361
  - postpone\_output (*http*), 362
  - preread\_buffer\_size (*stream*), 456
  - preread\_timeout (*stream*), 456
  - prometheus (*http*), 184
  - prometheus\_template (*http*), 184
  - protocol, 485
  - proxy\_bind (*http*), 187
  - proxy\_bind (*stream*), 406
  - proxy\_buffer, 470
  - proxy\_buffer\_size (*http*), 187
  - proxy\_buffer\_size (*stream*), 406
  - proxy\_buffering (*http*), 188
  - proxy\_buffers (*http*), 188
  - proxy\_busy\_buffers\_size (*http*), 188
  - proxy\_cache (*http*), 189
  - proxy\_cache\_background\_update (*http*), 190
  - proxy\_cache\_bypass (*http*), 190
  - proxy\_cache\_convert\_head (*http*), 190
  - proxy\_cache\_key (*http*), 191
  - proxy\_cache\_lock (*http*), 191
  - proxy\_cache\_lock\_age (*http*), 191
  - proxy\_cache\_lock\_timeout (*http*), 192
  - proxy\_cache\_max\_range\_offset (*http*), 192
  - proxy\_cache\_methods (*http*), 192
  - proxy\_cache\_min\_uses (*http*), 192
  - proxy\_cache\_path (*http*), 193
  - proxy\_cache\_revalidate (*http*), 194
  - proxy\_cache\_use\_stale (*http*), 195
  - proxy\_cache\_valid (*http*), 195
  - proxy\_connect\_timeout (*http*), 196
  - proxy\_connect\_timeout (*stream*), 406
  - proxy\_connection\_drop (*http*), 197
  - proxy\_connection\_drop (*stream*), 407
  - proxy\_cookie\_domain (*http*), 197
  - proxy\_cookie\_flags (*http*), 198
  - proxy\_cookie\_path (*http*), 198
  - proxy\_download\_rate (*stream*), 407
  - proxy\_force\_ranges (*http*), 199
  - proxy\_half\_close (*stream*), 408
  - proxy\_headers\_hash\_bucket\_size (*http*), 199
  - proxy\_headers\_hash\_max\_size (*http*), 200
  - proxy\_hide\_header (*http*), 200
  - proxy\_http3\_hq (*http*), 200
  - proxy\_http3\_max\_concurrent\_streams (*http*), 201
  - proxy\_http3\_stream\_buffer\_size (*http*), 201
  - proxy\_http\_version (*http*), 200
  - proxy\_ignore\_client\_abort (*http*), 201
  - proxy\_ignore\_headers (*http*), 201
  - proxy\_intercept\_errors (*http*), 202
  - proxy\_limit\_rate (*http*), 202
  - proxy\_max\_temp\_file\_size (*http*), 203
  - proxy\_method (*http*), 203
  - proxy\_next\_upstream (*http*), 203
  - proxy\_next\_upstream (*stream*), 408
  - proxy\_next\_upstream\_timeout (*http*), 204
  - proxy\_next\_upstream\_timeout (*stream*), 408
  - proxy\_next\_upstream\_tries (*http*), 205
  - proxy\_next\_upstream\_tries (*stream*), 408
  - proxy\_no\_cache (*http*), 205

- proxy\_pass (*http*), 205
  - proxy\_pass (*stream*), 409
  - proxy\_pass\_error\_message, 470
  - proxy\_pass\_header (*http*), 207
  - proxy\_pass\_request\_body (*http*), 207
  - proxy\_pass\_request\_headers (*http*), 207
  - proxy\_protocol, 470
  - proxy\_protocol (*stream*), 409
  - proxy\_protocol\_timeout (*stream*), 456
  - proxy\_quic\_active\_connection\_id\_limit (*http*), 208
  - proxy\_quic\_gso (*http*), 208
  - proxy\_quic\_host\_key (*http*), 208
  - proxy\_read\_timeout (*http*), 209
  - proxy\_redirect (*http*), 209
  - proxy\_request\_buffering (*http*), 210
  - proxy\_requests (*stream*), 409
  - proxy\_responses (*stream*), 410
  - proxy\_send\_lowat (*http*), 211
  - proxy\_send\_timeout (*http*), 211
  - proxy\_set\_body (*http*), 211
  - proxy\_set\_header (*http*), 212
  - proxy\_smtp\_auth, 471
  - proxy\_socket\_keepalive (*http*), 212
  - proxy\_socket\_keepalive (*stream*), 410
  - proxy\_ssl (*stream*), 410
  - proxy\_ssl\_certificate (*http*), 213
  - proxy\_ssl\_certificate (*stream*), 411
  - proxy\_ssl\_certificate\_key (*http*), 213
  - proxy\_ssl\_certificate\_key (*stream*), 411
  - proxy\_ssl\_ciphers (*http*), 214
  - proxy\_ssl\_ciphers (*stream*), 412
  - proxy\_ssl\_conf\_command (*http*), 214
  - proxy\_ssl\_conf\_command (*stream*), 412
  - proxy\_ssl\_crl (*http*), 215
  - proxy\_ssl\_crl (*stream*), 412
  - proxy\_ssl\_name (*http*), 215
  - proxy\_ssl\_name (*stream*), 413
  - proxy\_ssl\_ntls (*http*), 215
  - proxy\_ssl\_ntls (*stream*), 413
  - proxy\_ssl\_password\_file (*http*), 216
  - proxy\_ssl\_password\_file (*stream*), 414
  - proxy\_ssl\_protocols (*http*), 216
  - proxy\_ssl\_protocols (*stream*), 414
  - proxy\_ssl\_server\_name (*http*), 217
  - proxy\_ssl\_server\_name (*stream*), 414
  - proxy\_ssl\_session\_reuse (*http*), 217
  - proxy\_ssl\_session\_reuse (*stream*), 414
  - proxy\_ssl\_trusted\_certificate (*http*), 217
  - proxy\_ssl\_trusted\_certificate (*stream*), 415
  - proxy\_ssl\_verify (*http*), 217
  - proxy\_ssl\_verify (*stream*), 415
  - proxy\_ssl\_verify\_depth (*http*), 218
  - proxy\_ssl\_verify\_depth (*stream*), 415
  - proxy\_store (*http*), 218
  - proxy\_store\_access (*http*), 219
  - proxy\_temp\_file\_write\_size (*http*), 220
  - proxy\_temp\_path (*http*), 220
  - proxy\_timeout, 471
  - proxy\_timeout (*stream*), 415
  - proxy\_upload\_rate (*stream*), 416
- ## Q
- queue (*http*), 287
  - quic\_active\_connection\_id\_limit (*http*), 332
  - quic\_bpf (*http*), 332
  - quic\_gso (*http*), 333
  - quic\_host\_key (*http*), 333
  - quic\_retry (*http*), 333
- ## R
- random (*http*), 288
  - random (*stream*), 443
  - random\_index (*http*), 221
  - rdp\_preread (*stream*), 417
  - read\_ahead (*http*), 362
  - real\_ip\_header (*http*), 222
  - real\_ip\_recursive (*http*), 222
  - recursive\_error\_pages (*http*), 362
  - referer\_hash\_bucket\_size (*http*), 224
  - referer\_hash\_max\_size (*http*), 224
  - request\_pool\_size (*http*), 362
  - reset\_timedout\_connection (*http*), 363
  - resolver, 486
  - resolver (*http*), 363
  - resolver (*stream*), 456
  - resolver\_timeout, 487
  - resolver\_timeout (*http*), 364
  - resolver\_timeout (*stream*), 457
  - response\_time\_factor (*http*), 288

response\_time\_factor (*stream*), 444  
 return (*http*), 227  
 return (*stream*), 419  
 rewrite (*http*), 227  
 rewrite\_log (*http*), 228  
 root (*http*), 364

## S

satisfy (*http*), 365  
 scgi\_bind (*http*), 230  
 scgi\_buffer\_size (*http*), 231  
 scgi\_buffering (*http*), 231  
 scgi\_buffers (*http*), 231  
 scgi\_busy\_buffers\_size (*http*), 232  
 scgi\_cache (*http*), 232  
 scgi\_cache\_background\_update (*http*), 232  
 scgi\_cache\_bypass (*http*), 233  
 scgi\_cache\_key (*http*), 233  
 scgi\_cache\_lock (*http*), 233  
 scgi\_cache\_lock\_age (*http*), 234  
 scgi\_cache\_lock\_timeout (*http*), 234  
 scgi\_cache\_max\_range\_offset (*http*), 234  
 scgi\_cache\_methods (*http*), 234  
 scgi\_cache\_min\_uses (*http*), 235  
 scgi\_cache\_path (*http*), 235  
 scgi\_cache\_revalidate (*http*), 236  
 scgi\_cache\_use\_stale (*http*), 236  
 scgi\_cache\_valid (*http*), 237  
 scgi\_connect\_timeout (*http*), 238  
 scgi\_connection\_drop (*http*), 238  
 scgi\_force\_ranges (*http*), 239  
 scgi\_hide\_header (*http*), 239  
 scgi\_ignore\_client\_abort (*http*), 239  
 scgi\_ignore\_headers (*http*), 239  
 scgi\_intercept\_errors (*http*), 240  
 scgi\_limit\_rate (*http*), 240  
 scgi\_max\_temp\_file\_size (*http*), 240  
 scgi\_next\_upstream (*http*), 241  
 scgi\_next\_upstream\_timeout (*http*), 242  
 scgi\_next\_upstream\_tries (*http*), 242  
 scgi\_no\_cache (*http*), 242  
 scgi\_param (*http*), 243  
 scgi\_pass (*http*), 243  
 scgi\_pass\_header (*http*), 244  
 scgi\_pass\_request\_body (*http*), 244

scgi\_pass\_request\_headers (*http*), 244  
 scgi\_read\_timeout (*http*), 244  
 scgi\_request\_buffering (*http*), 245  
 scgi\_send\_timeout (*http*), 245  
 scgi\_socket\_keepalive (*http*), 245  
 scgi\_store (*http*), 246  
 scgi\_store\_access (*http*), 247  
 scgi\_temp\_file\_write\_size (*http*), 247  
 scgi\_temp\_path (*http*), 247  
 secure\_link (*http*), 248  
 secure\_link\_md5 (*http*), 249  
 secure\_link\_secret (*http*), 249  
 send\_lowat (*http*), 365  
 send\_timeout (*http*), 365  
 sendfile (*http*), 365  
 sendfile\_max\_chunk (*http*), 366  
 server, 487  
 server (*http*), 289, 366  
 server (*stream*), 438, 457  
 server\_name, 487  
 server\_name (*http*), 366  
 server\_name (*stream*), 458  
 server\_name\_in\_redirect (*http*), 368  
 server\_names\_hash\_bucket\_size (*http*), 369  
 server\_names\_hash\_bucket\_size (*stream*), 459  
 server\_names\_hash\_max\_size (*http*), 369  
 server\_names\_hash\_max\_size (*stream*), 459  
 server\_tokens (*http*), 369  
 set (*http*), 228  
 set (*stream*), 420  
 set\_real\_ip\_from, 472  
 set\_real\_ip\_from (*http*), 222  
 set\_real\_ip\_from (*stream*), 418  
 slice (*http*), 251  
 smtp\_auth, 472  
 smtp\_capabilities, 473  
 smtp\_client\_buffer, 473  
 smtp\_greeting\_delay, 473  
 source\_charset (*http*), 78  
 split\_clients (*http*), 252  
 split\_clients (*stream*), 421  
 ssi (*http*), 253  
 ssi\_last\_modified (*http*), 253  
 ssi\_min\_file\_chunk (*http*), 254  
 ssi\_silent\_errors (*http*), 254

ssi\_types (*http*), 254  
 ssi\_value\_length (*http*), 254  
 ssl\_alpn (*stream*), 422  
 ssl\_buffer\_size (*http*), 260  
 ssl\_certificate, 475  
 ssl\_certificate (*http*), 260  
 ssl\_certificate (*stream*), 423  
 ssl\_certificate\_key, 475  
 ssl\_certificate\_key (*http*), 262  
 ssl\_certificate\_key (*stream*), 424  
 ssl\_ciphers, 476  
 ssl\_ciphers (*http*), 262  
 ssl\_ciphers (*stream*), 424  
 ssl\_client\_certificate, 476  
 ssl\_client\_certificate (*http*), 263  
 ssl\_client\_certificate (*stream*), 425  
 ssl\_conf\_command, 476  
 ssl\_conf\_command (*http*), 263  
 ssl\_conf\_command (*stream*), 425  
 ssl\_crl, 477  
 ssl\_crl (*http*), 264  
 ssl\_crl (*stream*), 426  
 ssl\_dhparam, 477  
 ssl\_dhparam (*http*), 264  
 ssl\_dhparam (*stream*), 426  
 ssl\_early\_data (*http*), 264  
 ssl\_ecdh\_curve, 478  
 ssl\_ecdh\_curve (*http*), 264  
 ssl\_ecdh\_curve (*stream*), 426  
 ssl\_engine (*core*), 23  
 ssl\_handshake\_timeout (*stream*), 427  
 ssl\_ntls (*http*), 265  
 ssl\_ntls (*stream*), 427  
 ssl\_ocsp (*http*), 266  
 ssl\_ocsp\_cache (*http*), 266  
 ssl\_ocsp\_responder (*http*), 266  
 ssl\_password\_file, 478  
 ssl\_password\_file (*http*), 267  
 ssl\_password\_file (*stream*), 427  
 ssl\_prefer\_server\_ciphers, 479  
 ssl\_prefer\_server\_ciphers (*http*), 267  
 ssl\_prefer\_server\_ciphers (*stream*), 428  
 ssl\_preread (*stream*), 436  
 ssl\_protocols, 479  
 ssl\_protocols (*http*), 267  
 ssl\_protocols (*stream*), 428  
 ssl\_reject\_handshake (*http*), 268  
 ssl\_session\_cache, 479  
 ssl\_session\_cache (*http*), 268  
 ssl\_session\_cache (*stream*), 429  
 ssl\_session\_ticket\_key, 480  
 ssl\_session\_ticket\_key (*http*), 269  
 ssl\_session\_ticket\_key (*stream*), 429  
 ssl\_session\_tickets, 480  
 ssl\_session\_tickets (*http*), 269  
 ssl\_session\_tickets (*stream*), 430  
 ssl\_session\_timeout, 481  
 ssl\_session\_timeout (*http*), 270  
 ssl\_session\_timeout (*stream*), 430  
 ssl\_stapling (*http*), 270  
 ssl\_stapling\_file (*http*), 270  
 ssl\_stapling\_responder (*http*), 271  
 ssl\_stapling\_verify (*http*), 271  
 ssl\_trusted\_certificate, 481  
 ssl\_trusted\_certificate (*http*), 271  
 ssl\_trusted\_certificate (*stream*), 430  
 ssl\_verify\_client, 481  
 ssl\_verify\_client (*http*), 272  
 ssl\_verify\_client (*stream*), 431  
 ssl\_verify\_depth, 481  
 ssl\_verify\_depth (*http*), 272  
 ssl\_verify\_depth (*stream*), 431  
 starttls, 482  
 state (*http*), 291  
 state (*stream*), 440  
 status\_zone (*http*), 370  
 status\_zone (*stream*), 460  
 sticky (*http*), 292  
 sticky (*stream*), 444  
 sticky\_secret (*http*), 294  
 sticky\_secret (*stream*), 447  
 sticky\_strict (*http*), 294  
 sticky\_strict (*stream*), 446  
 stream (*stream*), 460  
 stub\_status (*http*), 276  
 sub\_filter (*http*), 278  
 sub\_filter\_last\_modified (*http*), 279  
 sub\_filter\_once (*http*), 279  
 sub\_filter\_types (*http*), 279  
 subrequest\_output\_buffer\_size (*http*), 370

## T

tcp\_nodelay (*http*), 370  
 tcp\_nodelay (*stream*), 460  
 tcp\_nopush (*http*), 371  
 thread\_pool (*core*), 23  
 timeout, 488  
 timer\_resolution (*core*), 24  
 try\_files (*http*), 371  
 types (*http*), 373  
 types\_hash\_bucket\_size (*http*), 374  
 types\_hash\_max\_size (*http*), 374

## U

underscores\_in\_headers (*http*), 374  
 uninitialized\_variable\_warn (*http*), 228  
 upstream (*http*), 295  
 upstream (*stream*), 437  
 upstream\_probe (*http*), 299  
 upstream\_probe (*stream*), 450  
 upstream\_probe\_timeout (*stream*), 416  
 use (*core*), 24  
 user (*core*), 24  
 userid (*http*), 302  
 userid\_domain (*http*), 302  
 userid\_expires (*http*), 302  
 userid\_flags (*http*), 303  
 userid\_mark (*http*), 303  
 userid\_name (*http*), 303  
 userid\_p3p (*http*), 303  
 userid\_path (*http*), 304  
 userid\_service (*http*), 304  
 uwsgi\_bind (*http*), 305  
 uwsgi\_buffer\_size (*http*), 306  
 uwsgi\_buffering (*http*), 306  
 uwsgi\_buffers (*http*), 306  
 uwsgi\_busy\_buffers\_size (*http*), 307  
 uwsgi\_cache (*http*), 307  
 uwsgi\_cache\_background\_update (*http*), 307  
 uwsgi\_cache\_bypass (*http*), 308  
 uwsgi\_cache\_key (*http*), 308  
 uwsgi\_cache\_lock (*http*), 308  
 uwsgi\_cache\_lock\_age (*http*), 309  
 uwsgi\_cache\_lock\_timeout (*http*), 309  
 uwsgi\_cache\_max\_range\_offset (*http*), 309  
 uwsgi\_cache\_methods (*http*), 309

uwsgi\_cache\_min\_uses (*http*), 310  
 uwsgi\_cache\_path (*http*), 310  
 uwsgi\_cache\_revalidate (*http*), 311  
 uwsgi\_cache\_use\_stale (*http*), 311  
 uwsgi\_cache\_valid (*http*), 312  
 uwsgi\_connect\_timeout (*http*), 313  
 uwsgi\_connection\_drop (*http*), 313  
 uwsgi\_force\_ranges (*http*), 314  
 uwsgi\_hide\_header (*http*), 314  
 uwsgi\_ignore\_client\_abort (*http*), 314  
 uwsgi\_ignore\_headers (*http*), 314  
 uwsgi\_intercept\_errors (*http*), 315  
 uwsgi\_limit\_rate (*http*), 315  
 uwsgi\_max\_temp\_file\_size (*http*), 315  
 uwsgi\_modifier1 (*http*), 316  
 uwsgi\_modifier2 (*http*), 316  
 uwsgi\_next\_upstream (*http*), 316  
 uwsgi\_next\_upstream\_timeout (*http*), 317  
 uwsgi\_next\_upstream\_tries (*http*), 318  
 uwsgi\_no\_cache (*http*), 318  
 uwsgi\_param (*http*), 318  
 uwsgi\_pass (*http*), 319  
 uwsgi\_pass\_header (*http*), 319  
 uwsgi\_pass\_request\_body (*http*), 320  
 uwsgi\_pass\_request\_headers (*http*), 320  
 uwsgi\_read\_timeout (*http*), 320  
 uwsgi\_request\_buffering (*http*), 320  
 uwsgi\_send\_timeout (*http*), 321  
 uwsgi\_socket\_keepalive (*http*), 321  
 uwsgi\_ssl\_certificate (*http*), 321  
 uwsgi\_ssl\_certificate\_key (*http*), 322  
 uwsgi\_ssl\_ciphers (*http*), 322  
 uwsgi\_ssl\_conf\_command (*http*), 322  
 uwsgi\_ssl\_crl (*http*), 323  
 uwsgi\_ssl\_name (*http*), 323  
 uwsgi\_ssl\_password\_file (*http*), 323  
 uwsgi\_ssl\_protocols (*http*), 323  
 uwsgi\_ssl\_server\_name (*http*), 324  
 uwsgi\_ssl\_session\_reuse (*http*), 324  
 uwsgi\_ssl\_trusted\_certificate (*http*), 324  
 uwsgi\_ssl\_verify (*http*), 324  
 uwsgi\_ssl\_verify\_depth (*http*), 325  
 uwsgi\_store (*http*), 325  
 uwsgi\_store\_access (*http*), 326  
 uwsgi\_temp\_file\_write\_size (*http*), 326

uwsgi\_temp\_path (*http*), 326

## V

valid\_referers (*http*), 224

variables\_hash\_bucket\_size (*http*), 374

variables\_hash\_bucket\_size (*stream*), 460

variables\_hash\_max\_size (*http*), 375

variables\_hash\_max\_size (*stream*), 461

## W

wamr\_global\_heap\_size, 489

wamr\_heap\_size, 489

wamr\_stack\_size, 489

wasm\_modules, 492

wasmtime\_enable\_wasi, 490

wasmtime\_stack\_size, 490

worker\_aio\_requests (*core*), 25

worker\_connections (*core*), 25

worker\_cpu\_affinity (*core*), 25

worker\_priority (*core*), 26

worker\_processes (*core*), 26

worker\_rlimit\_core (*core*), 27

worker\_rlimit\_nofile (*core*), 27

worker\_shutdown\_timeout (*core*), 27

working\_directory (*core*), 27

## X

xclient, 471

xml\_entities (*http*), 335

xslt\_last\_modified (*http*), 335

xslt\_param (*http*), 335

xslt\_string\_param (*http*), 336

xslt\_stylesheet (*http*), 336

xslt\_types (*http*), 337

## Z

zone (*http*), 295

zone (*stream*), 440